Modélisations de foules dans un contexte et un environnement donné

CRAHAY--BOUDOU Florent

SCEI : 48160 ; MPI 2022-2023

Sommaire

Introduction & rapport à la ville

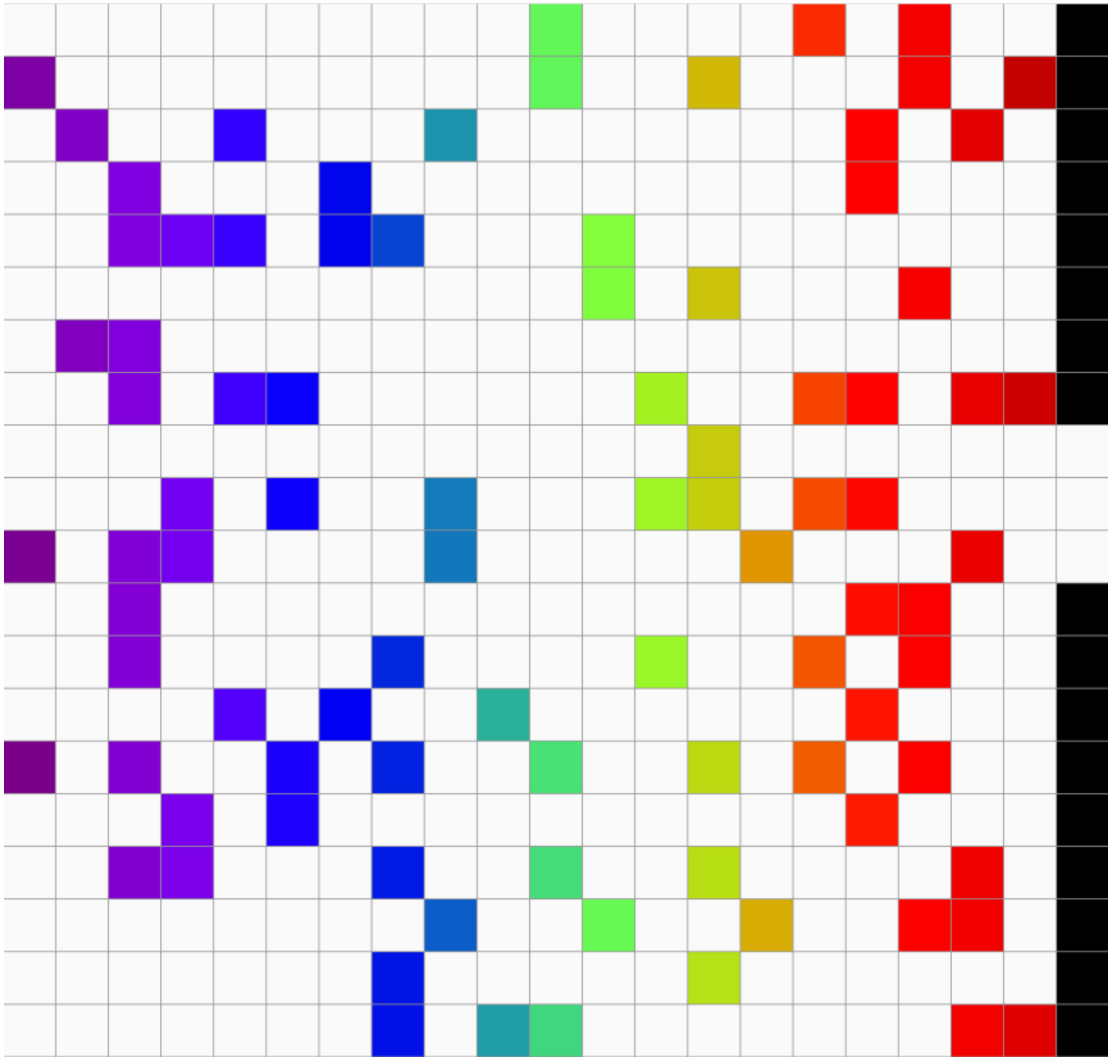
Permet de prévoir :

- comment se déroulera l’évacuation d’un futur bâtiment

- comment se déplaceront les personnes dans les espaces publics

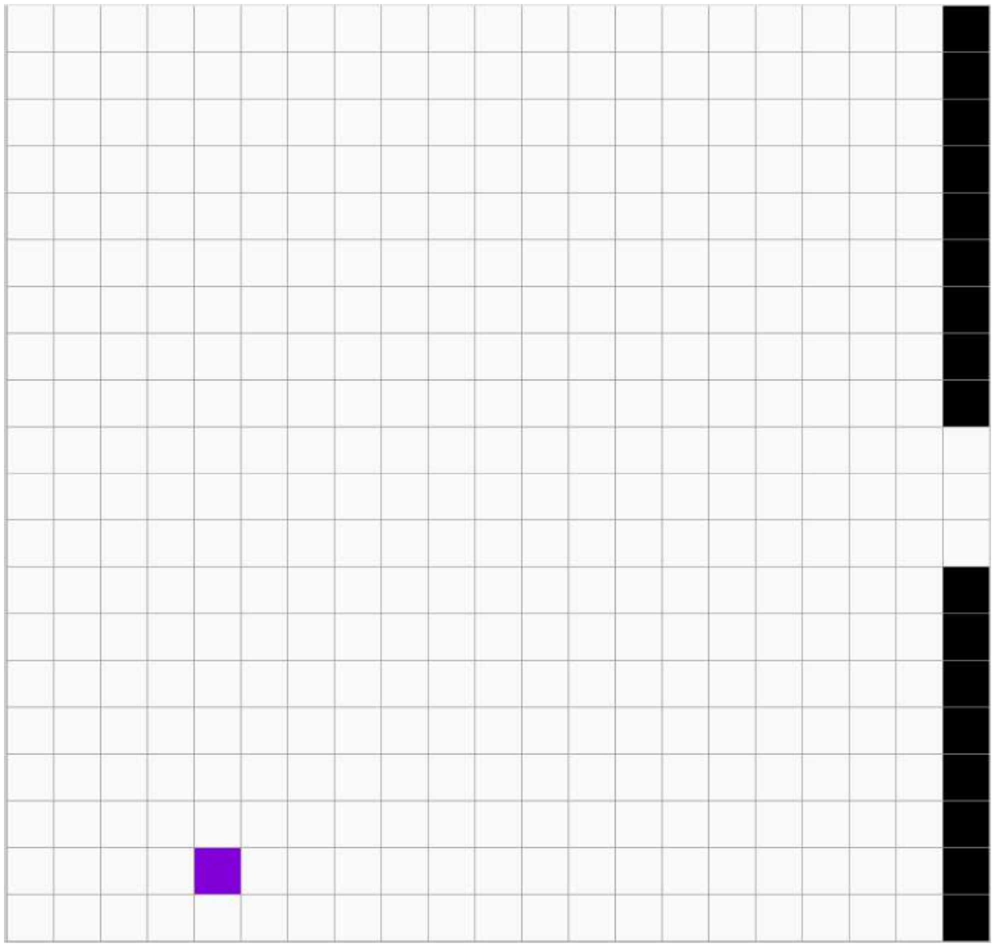
I-1) Modélisation multi-agent sur grille

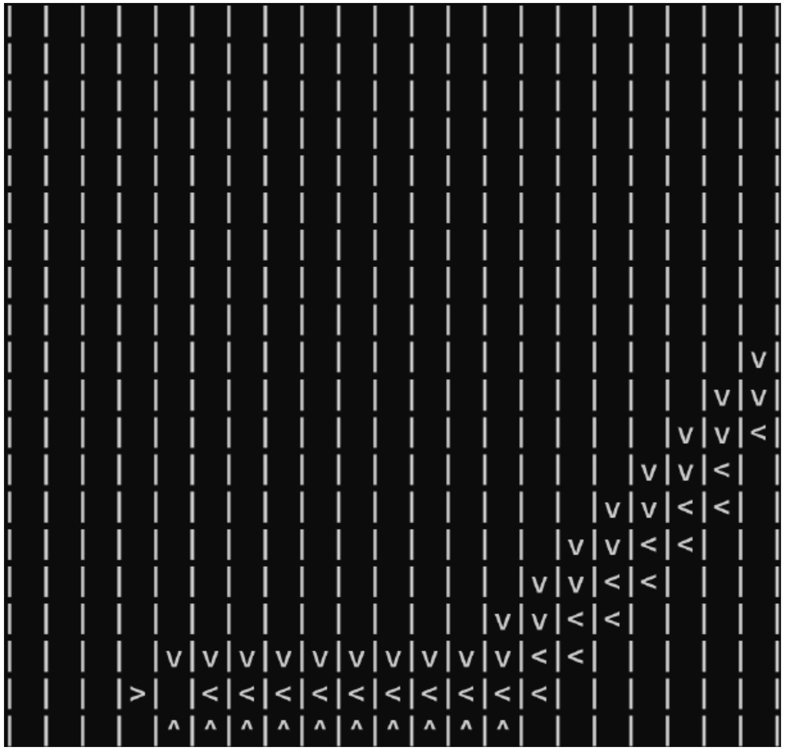
Présentation de la méthode



I-2) Programmation

Programme trouve_chemin

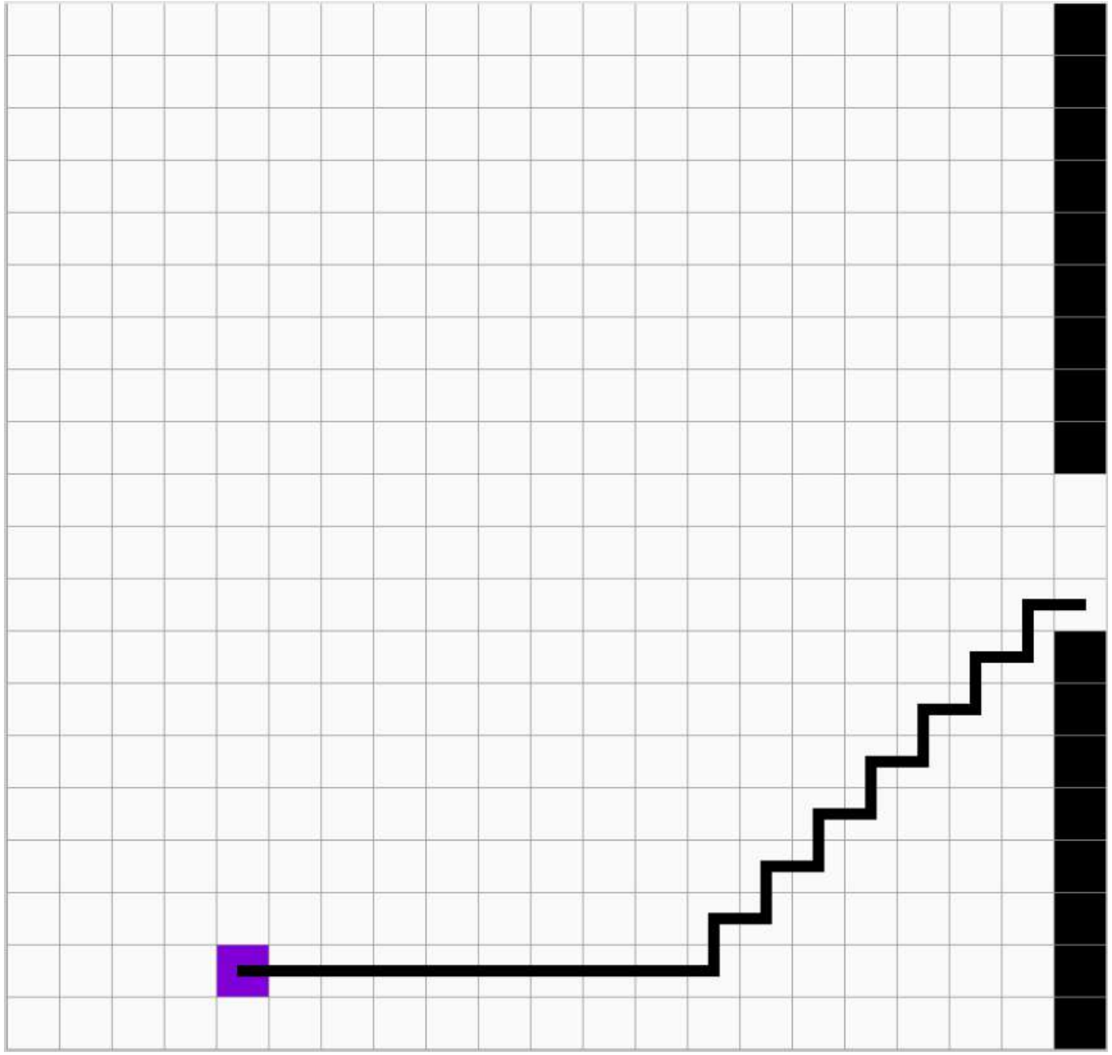




Utilisation de Best First

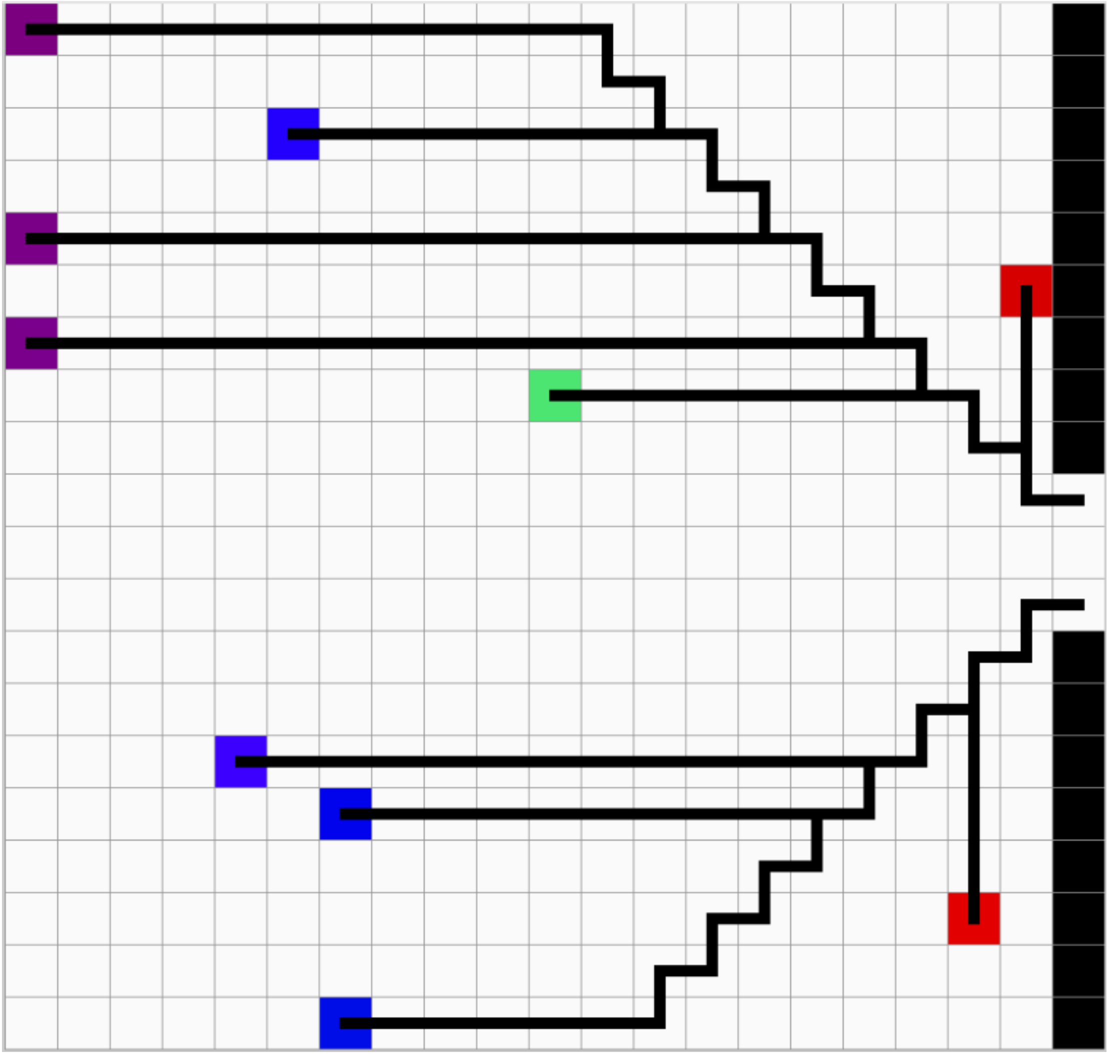
I-2) Programmation

Programme trouve_chemin



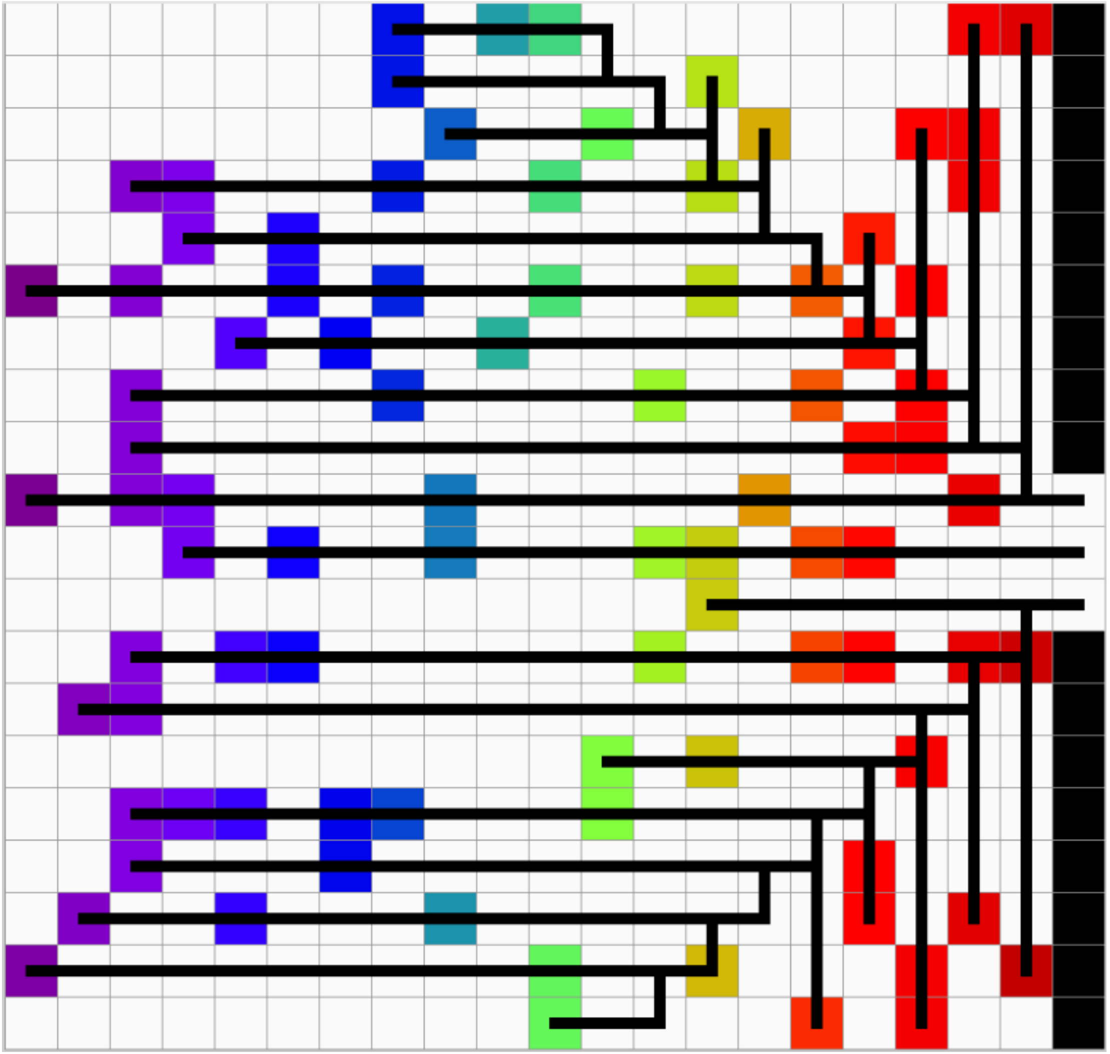
I-2) Programmation

Déroulé de l’algorithme



I-2) Programmation

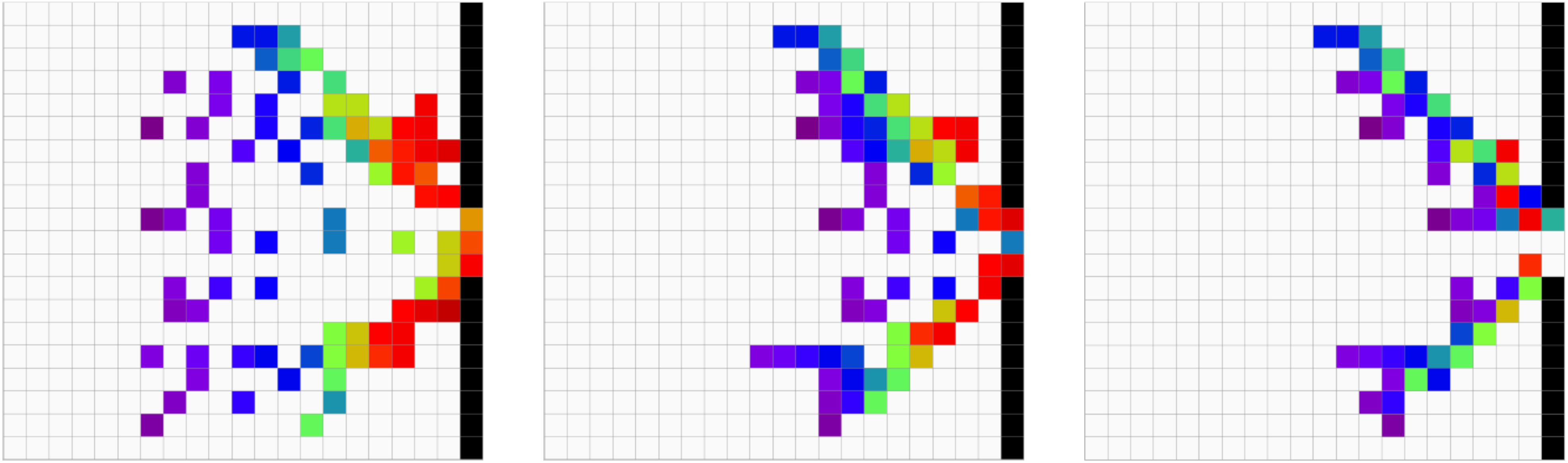
Déroulé de l'algorithme



I-2) Programmation

Réglage du défaut apparent

Embouteillage qui n’a pas lieu d’être

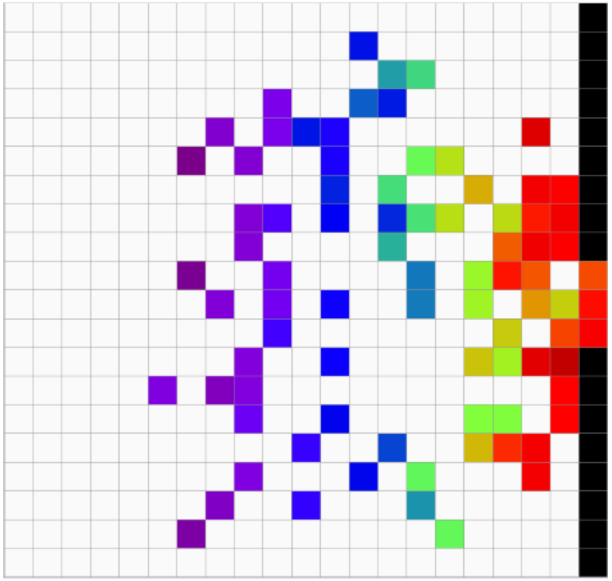


Grille au tour 6 | Grille au tour 12 | Grille au tour 20

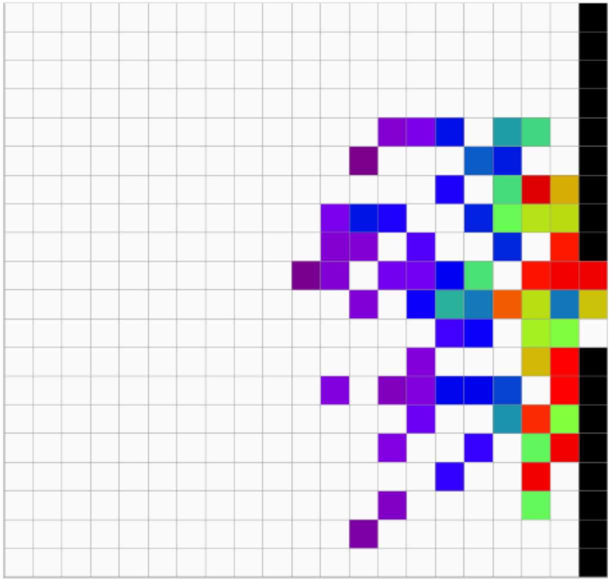
I-2) Programmation

Réglage du défaut apparent

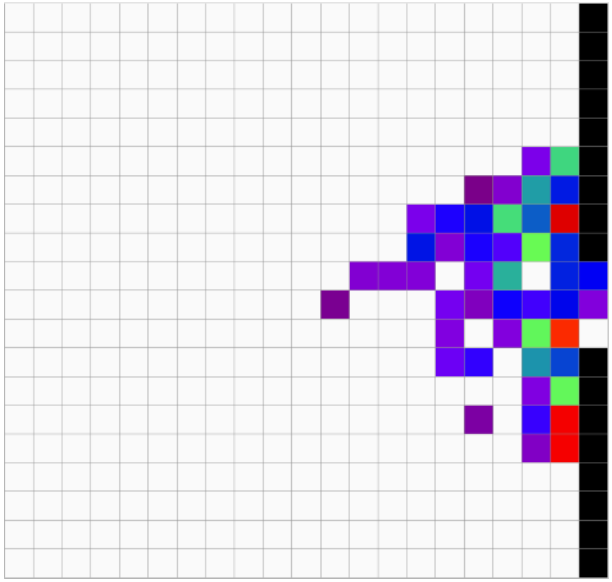
A* et recalcul



Grille au tour 6



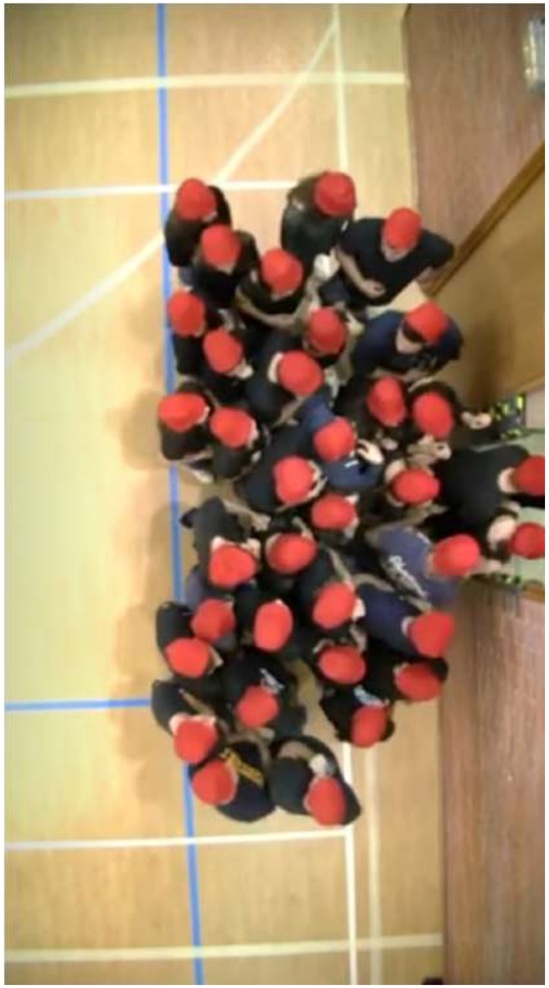
Grille au tour 12



Grille au tour 20

I-3) Confrontation avec le réel

Évacuation de 90 personnes à t = 20s



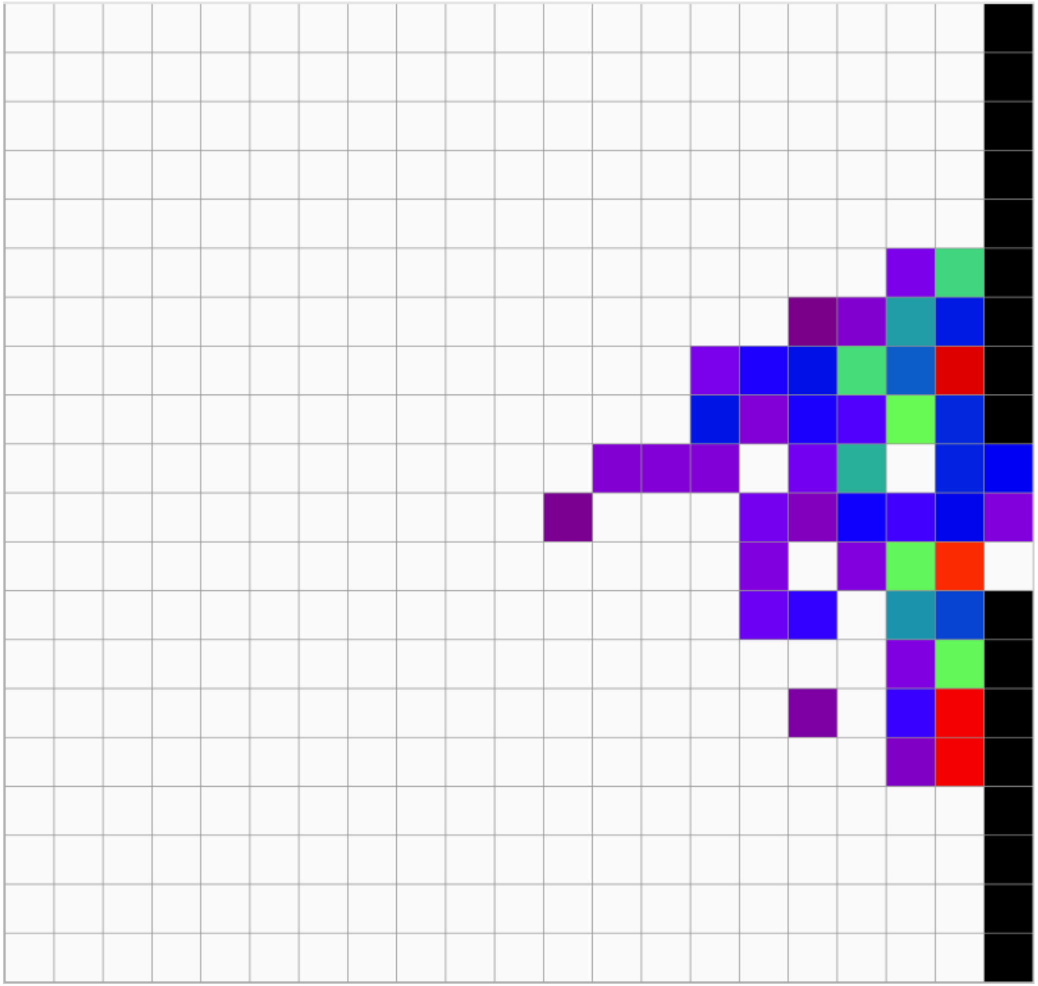
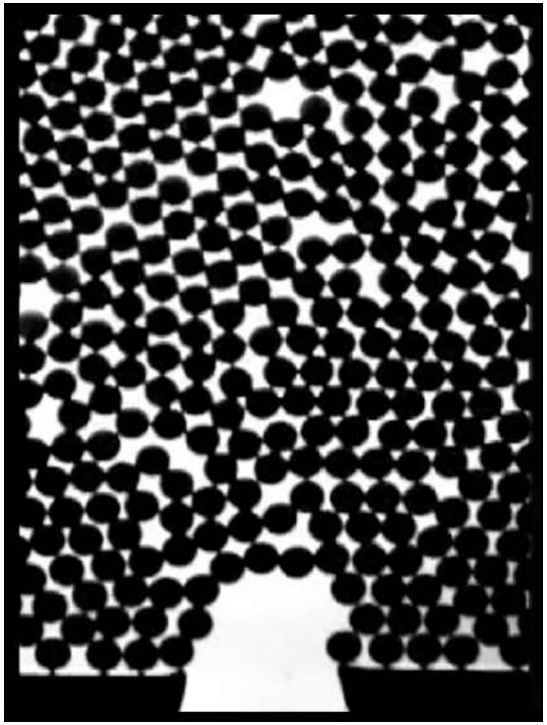


Image du papier Redefining the role of obstacles in pedestrian evacuation

Prise sur : https://iopscience.iop.org/article/10.1088/1367-2630/aaf4ca/meta#njpaaf4caf1

I-3) Confrontation avec le réel

Défaut du modèle : exemple du silo



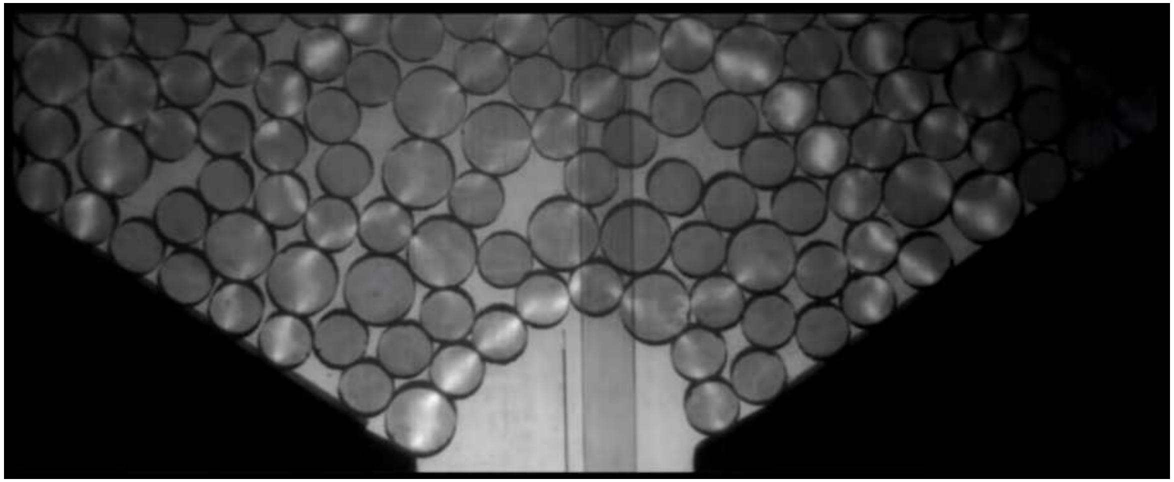
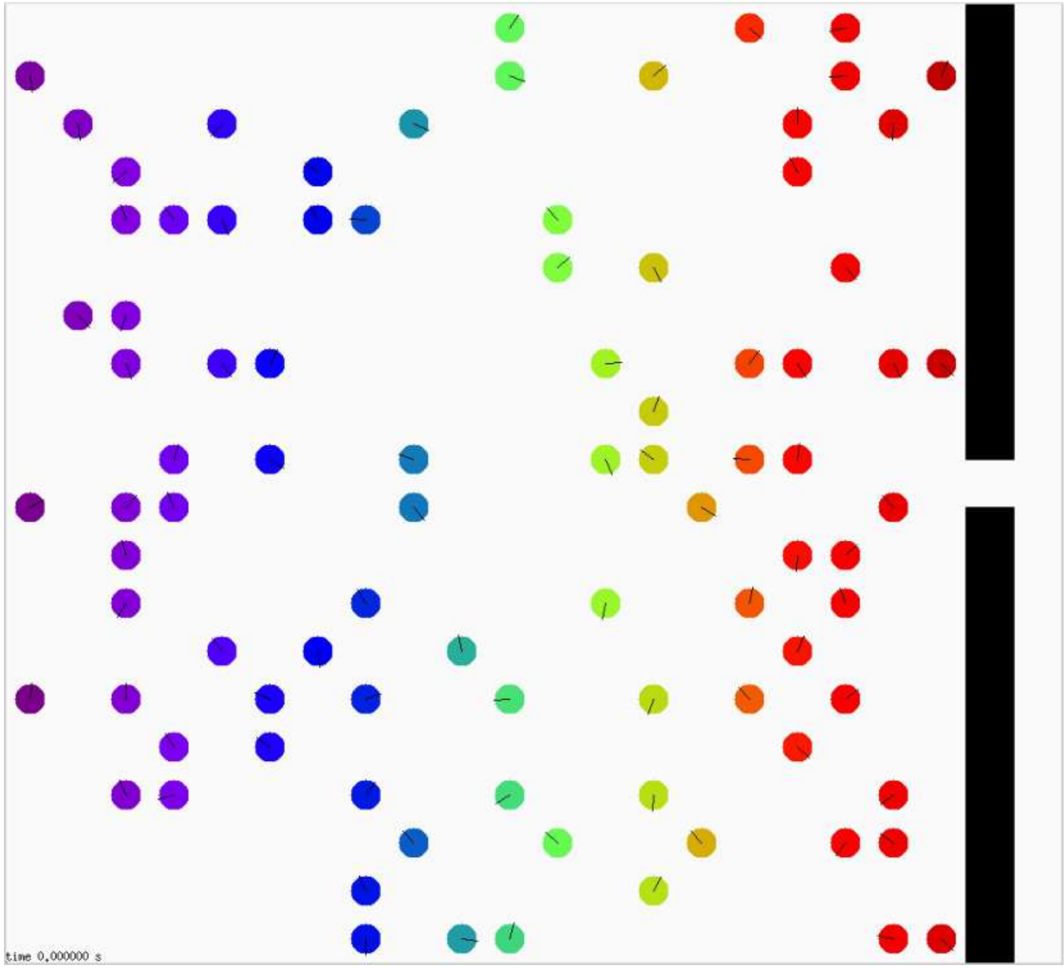


Image du papier Redefining the role of obstacles in pedestrian evacuation
Prise sur : https://iopscience.iop.org/article/10.1088/1367-2630/aaf4ca/meta#njpaaf4caf1

Tiré de « Le dilemme de l'évacuation » écrit par
Mehdi Moussaid, chercheur en science cognitive à l'institut Max Planck de Berlin

II-1) Modélisation en deux dimensions

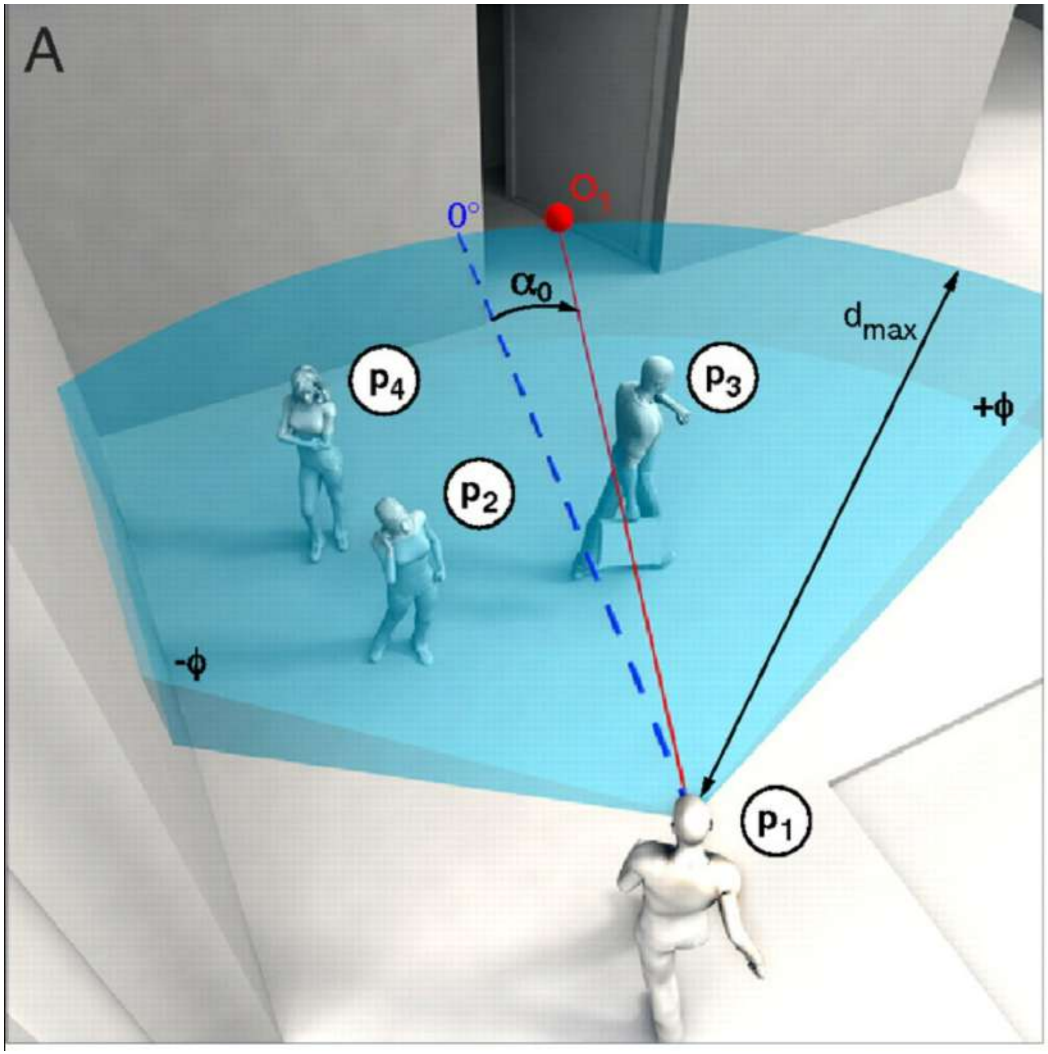
Présentation



How simple rules determine pedestrian behavior and crowd disasters
De Mehdi Moussaïd, Dirk Helbing, et Guy Theraulaz

II-2) Programmation

Méthode de choix de l’orientation

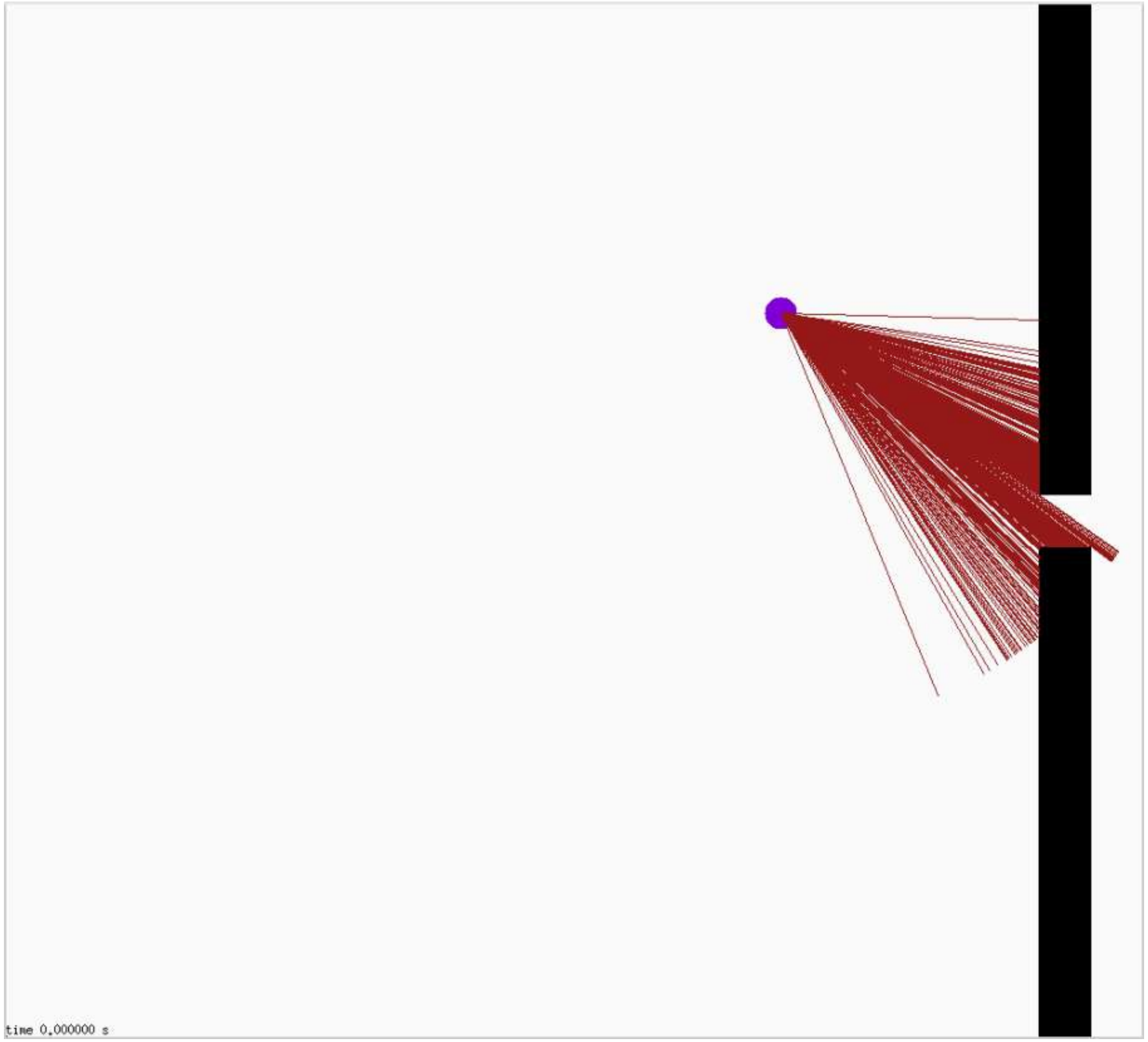


$d(\alpha_{des}) = \min(d(\alpha), \alpha \in [-\pi, \pi])$

$d(\alpha) = (d_{max})^2 + f(\alpha)^2 - 2d_{max}f(\alpha)\cos(\alpha_0 - \alpha)$

II-2) Programmation

Programmer une « vision » avec du raytracing



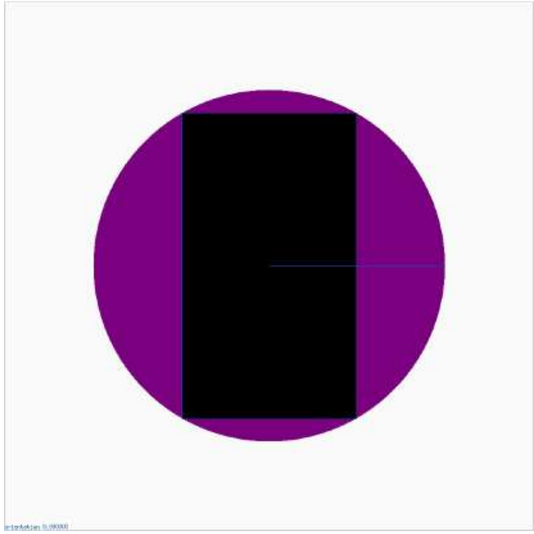
Box–Muller transformation

$$z = \sqrt{-2\ln(u_1)}\cos(2\pi u_2)$$

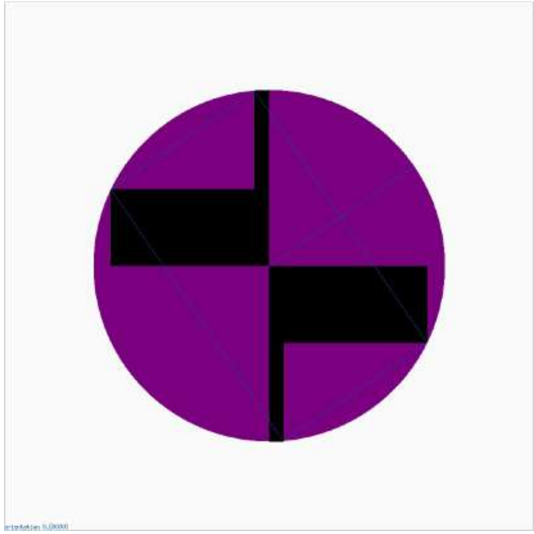
u_1, u_2, deux variables issues d'une fonction aléatoire uniforme sur [0; 1]

II-2) Programmation

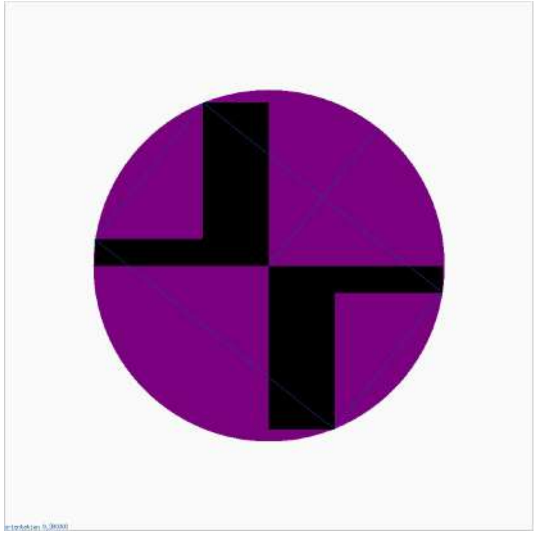
Découpage en rectangles suivant les axes



Pour direction = 0 rad



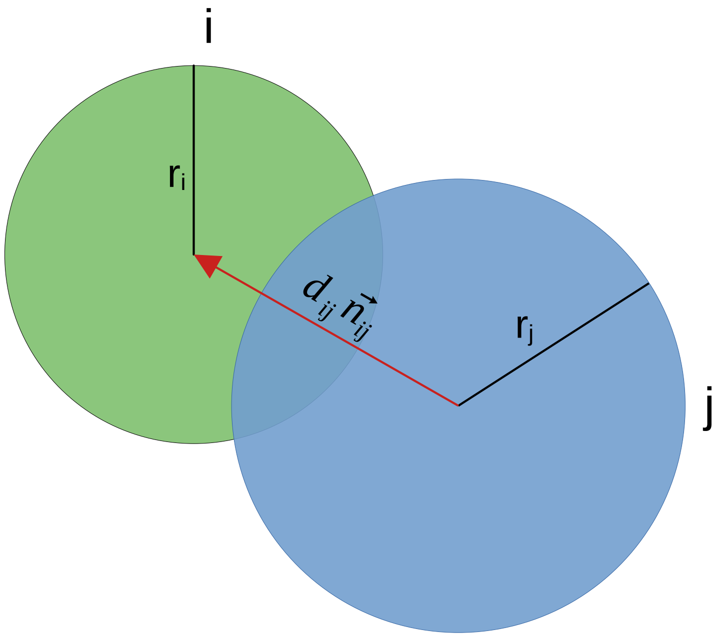
0,6 rad



0,9 rad

II-2) Programmation

Calcul du vecteur vitesse choisi



$$\vec{f}_{ij} = kg(r_i + r_j - d_{ij})\vec{n}_{ij}$$

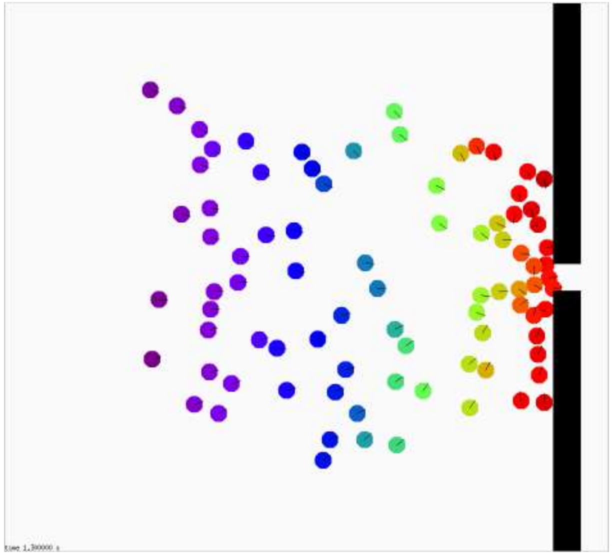
$$\vec{f}_{iW} = kg(r_i - d_{iW})\vec{n}_{iW}$$

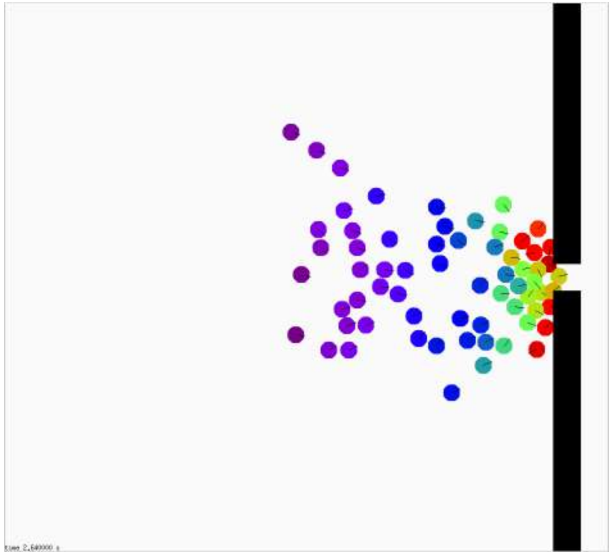
$$\vec{v}_{des}(t) = min(v_i^0, d_h)\cdot(u_x \cos(\alpha_{des}) + u_y \sin(\alpha_{des}))$$

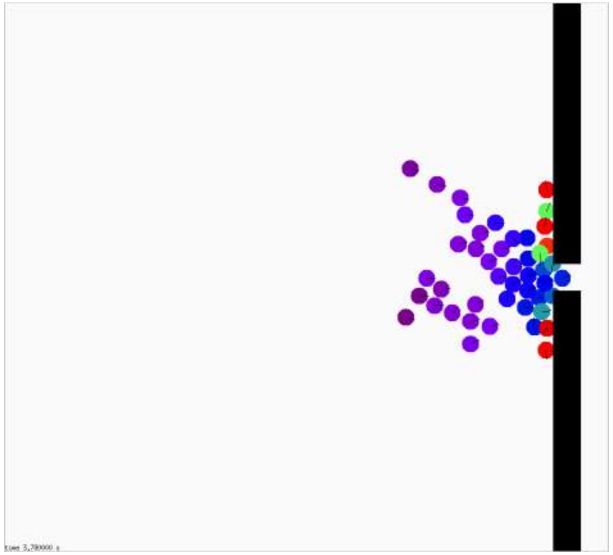
$$\frac{d\,\vec{v}_{des}}{dt} = (\vec{v}_{des} - \vec{v}_i) + \sum_{j, j \neq i}^{n} \vec{f}_{ij} / m_i + \sum_{W}^{m} \vec{f}_{iW} / m_i$$

II-3) Comparaison avec le réel

Déroulement de l'évacuation

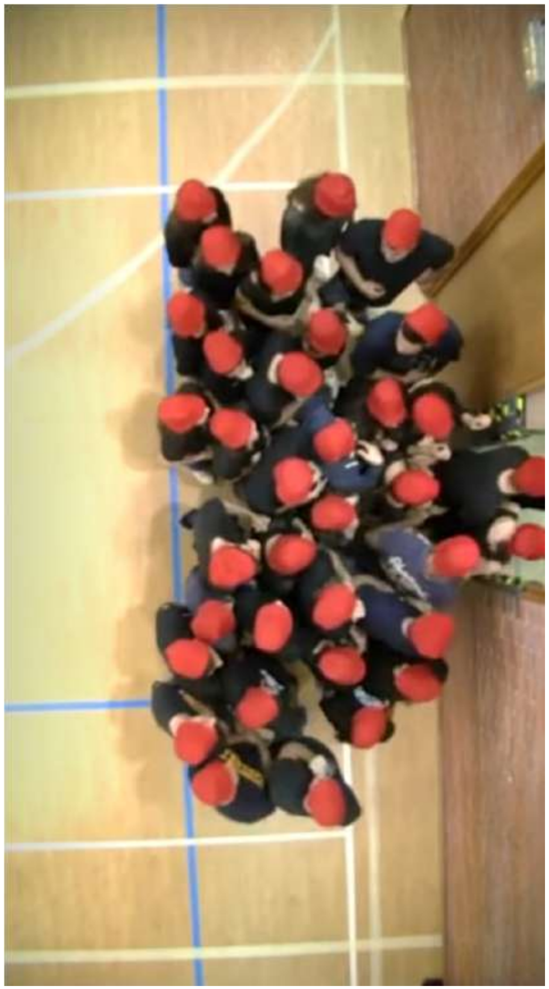






II-3) Comparaison avec le réel

Déroulement de l’évacuation



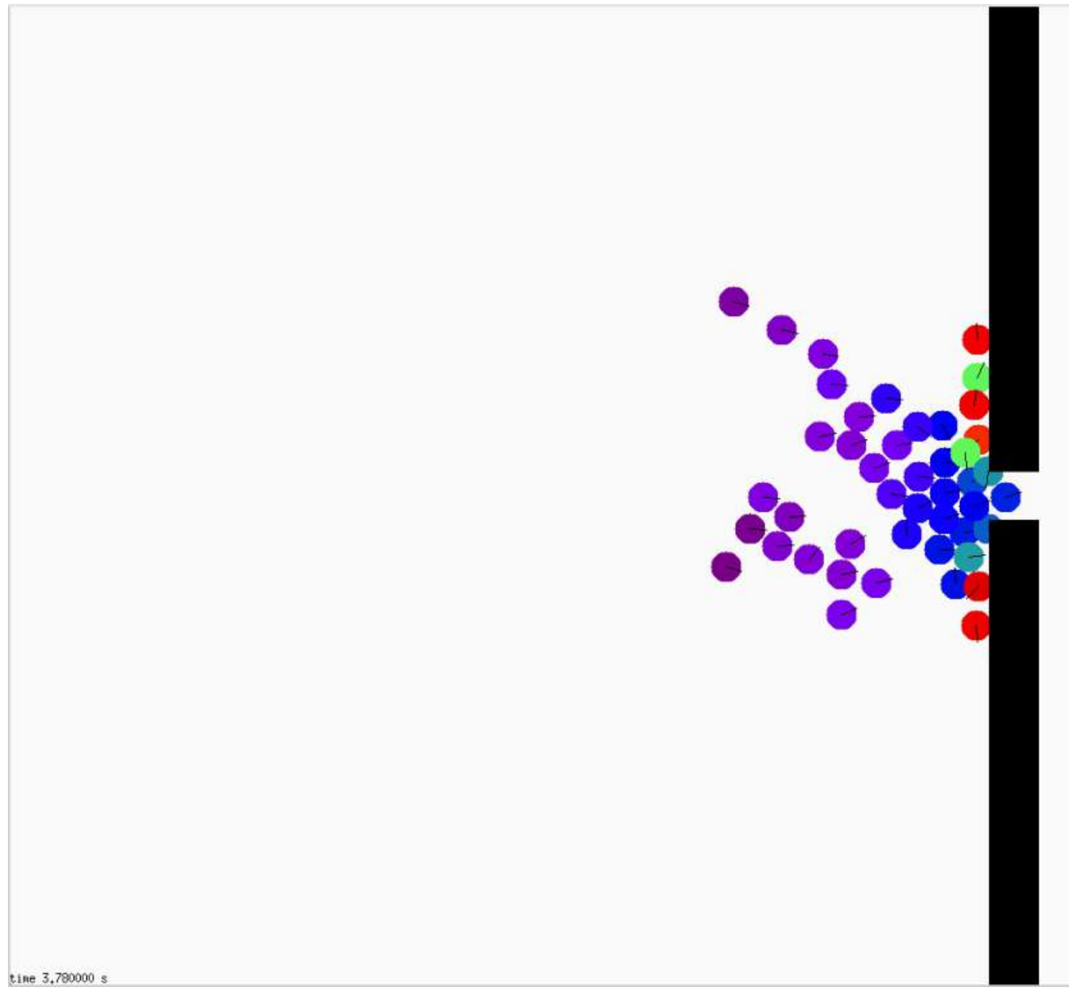


Image du papier Redefining the role of obstacles in pedestrian evacuation

Prise sur : https://iopscience.iop.org/article/10.1088/1367-2630/aaf4ca/meta#njpaaf4caf1

Conclusion

Modélisation multi-agents sur grille :

Qualités :

- Rapide (O(nm) ou n = nombre de personnes,
  m = taille du chemin le plus long)

Défauts :

- S’approche du modèle réel seulement lors des situations non-urgentes

Modélisation en deux dimensions :

Qualités :

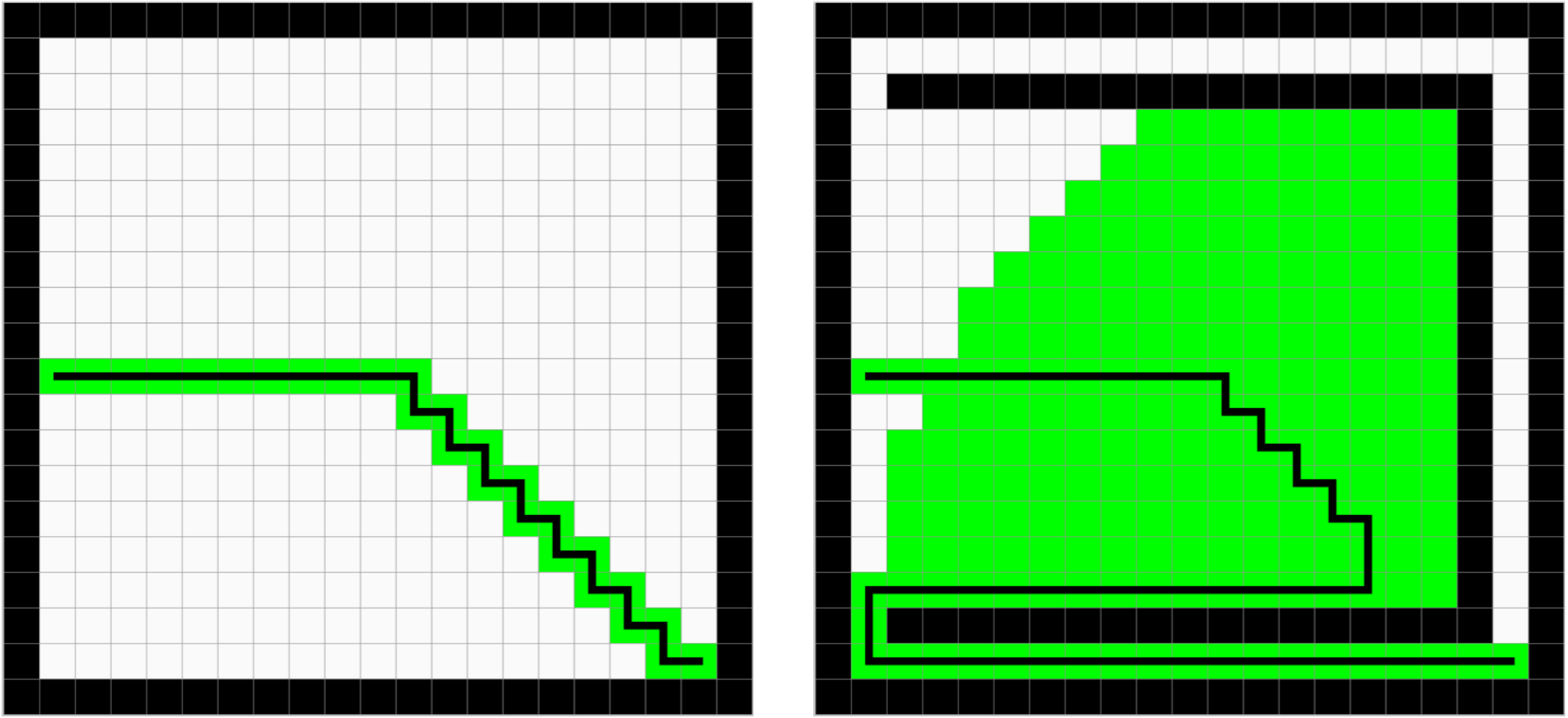
- Une meilleure modélisation

Défauts :

- Bien plus lourde en complexités temporelle ($O(n^2+nm)$ avec n = le nombre de personnes et m le nombre d’obstacles)

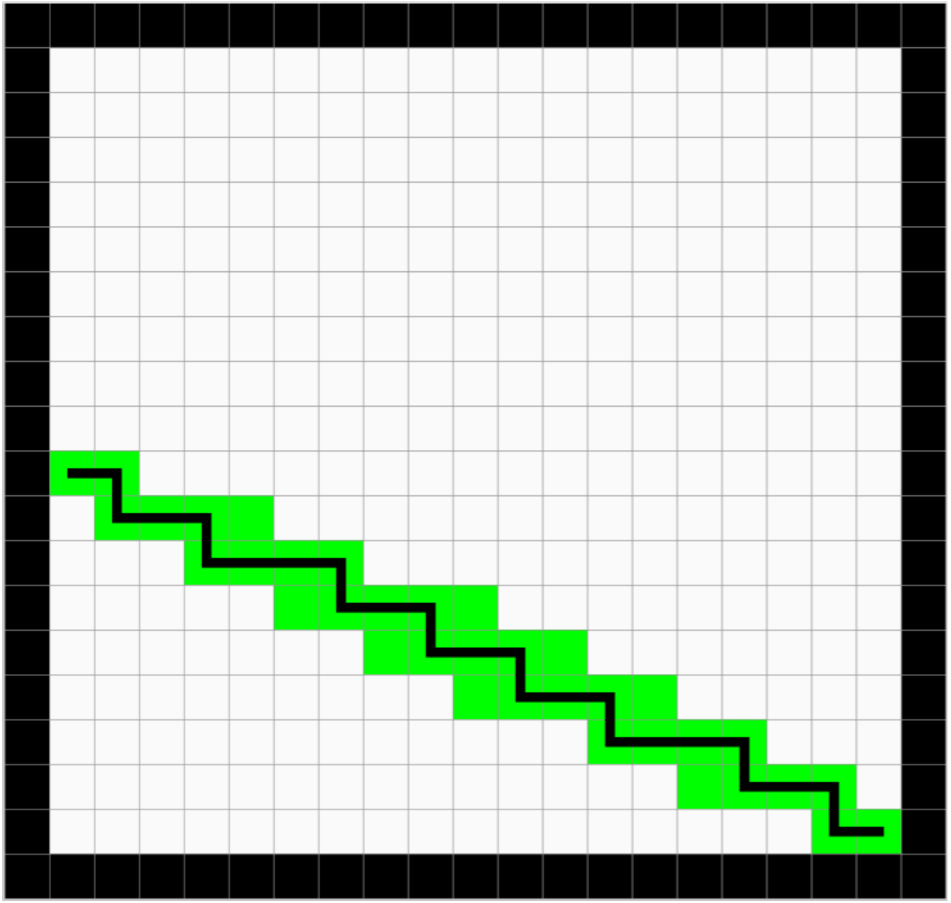
Annexe

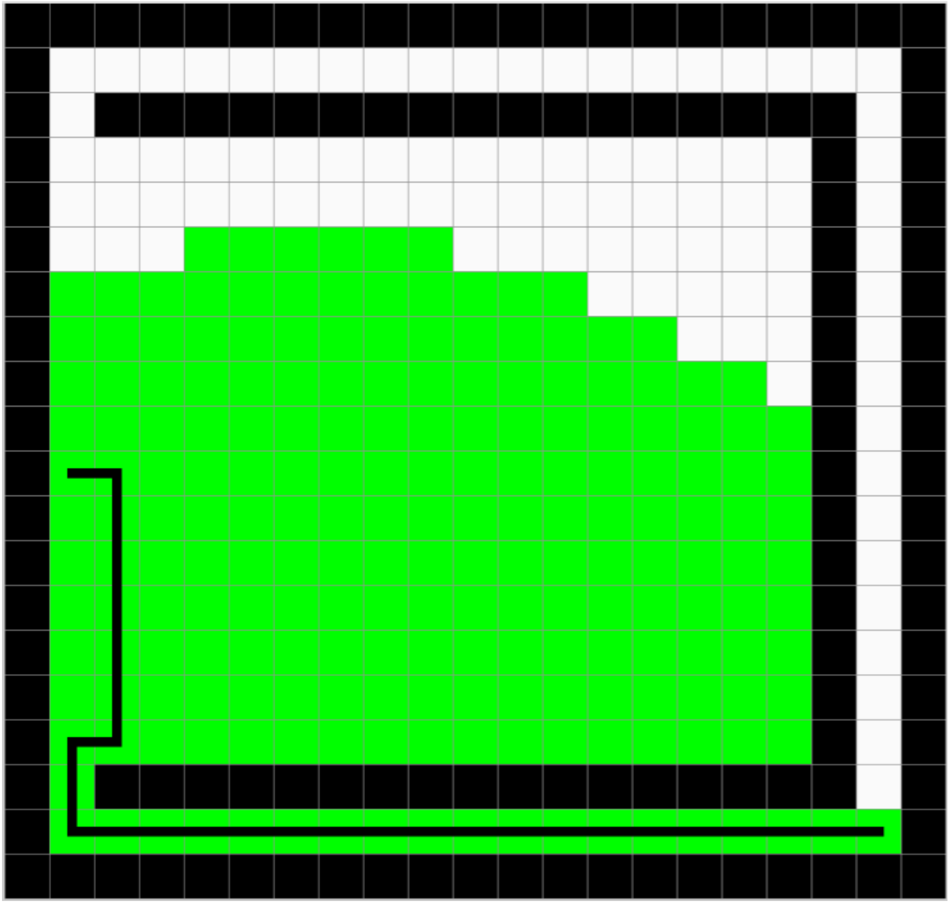
Dijkstra pondéré



Annexe

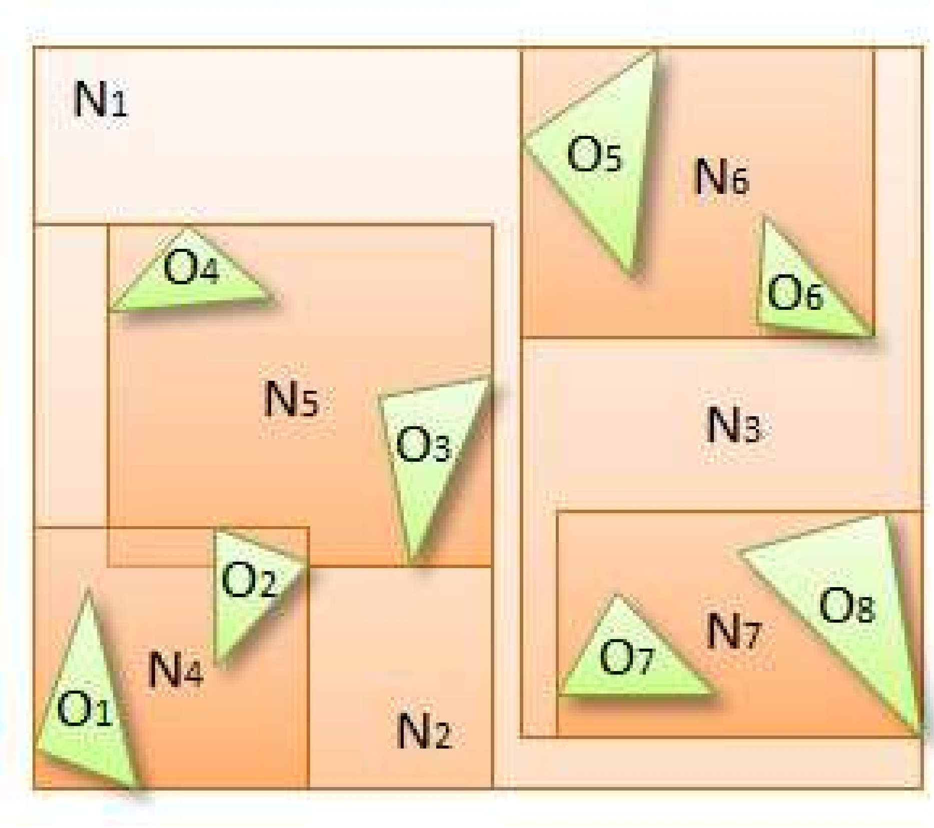
A*





Annexe

Bvh_tree



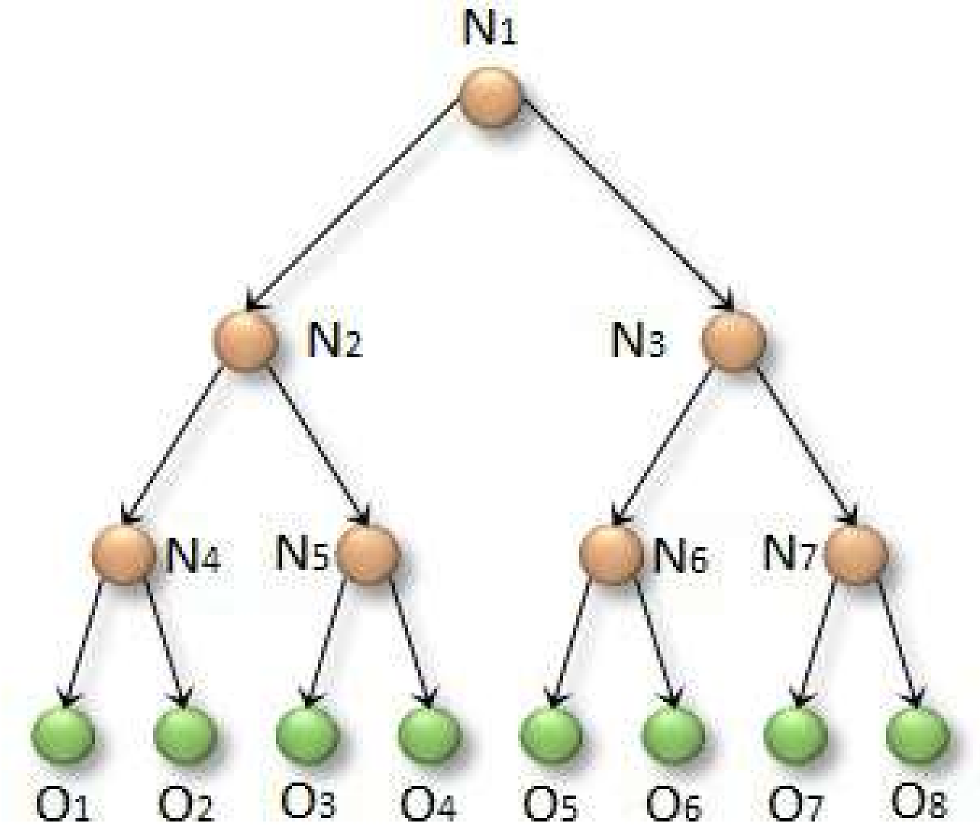


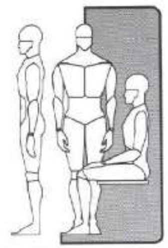
Image venant de https://developer.nvidia.com/blog/thinking-parallel-part-ii-tree-traversal-gpu/
écrit par Tero Karras

Annexe

Dimensions moyenne d'un homme

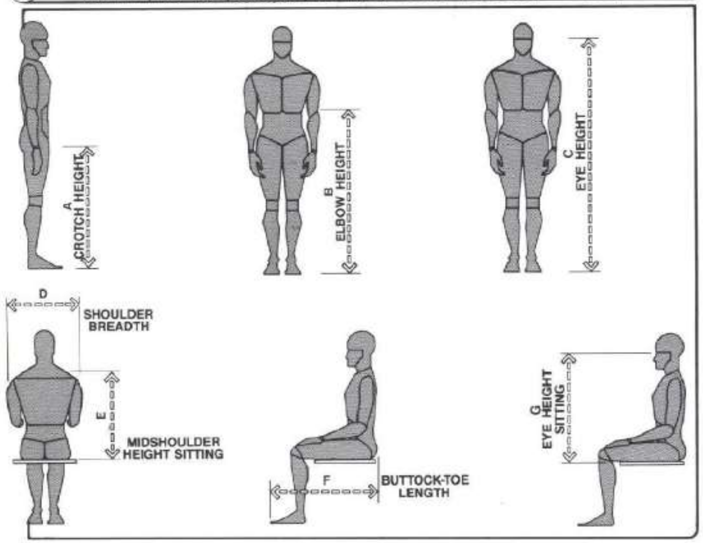
3

MISCELLANEOUS STRUCTURAL BODY DIMENSIONS



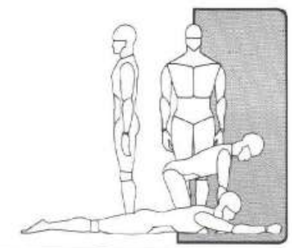
Adult Male and Female Miscellaneous Structural Body Dimensions in Inches and Centimeters by Age and Selected Percentiles

		A		B		C		D		E		F		G	
		in	cm	in	cm	in	cm	in	cm	in	cm	in	cm	in	cm
95	MEN	36.2	91.9	47.3	120.1	68.6	174.2	20.7	52.6	27.3	69.3	37.0	94.0	33.9	86.1
	WOMEN	32.0	81.3	43.6	110.7	64.1	162.8	17.0	43.2	24.6	62.5	37.0	94.0	31.7	80.5
5	MEN	30.8	78.2	41.3	104.9	60.8	154.4	17.4	44.2	23.7	60.2	32.0	81.3	30.0	76.2
	WOMEN	26.8	68.1	38.6	98.0	56.3	143.0	14.9	37.8	21.2	53.8	27.0	68.6	28.1	71.4



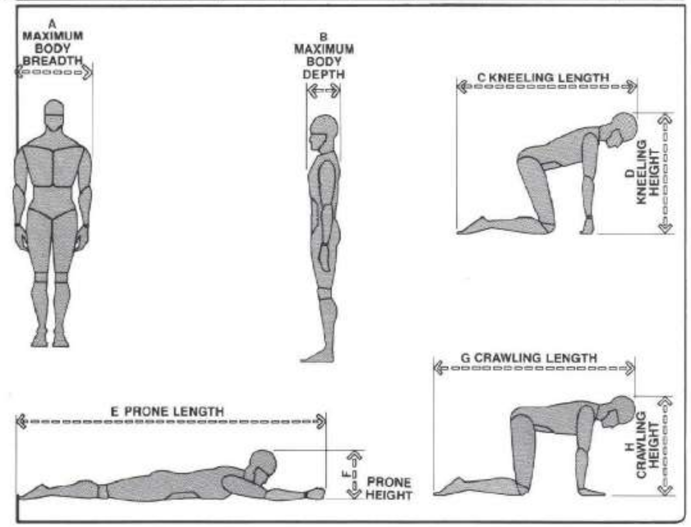
6

WORKING POSITIONS



Adult Male Working Positions in Inches and Centimeters and by Selected Percentiles*

		A	B	C	D	E	F	G	H
95	in	22.8	13.0	48.1	34.5	95.8	16.4	58.2	30.5
	cm	57.9	33.0	122.2	87.6	243.3	41.7	147.8	77.5
5	in	18.8	10.1	37.6	29.7	84.7	12.3	49.3	26.2
	cm	47.8	25.7	95.5	75.4	215.1	31.2	125.2	66.5

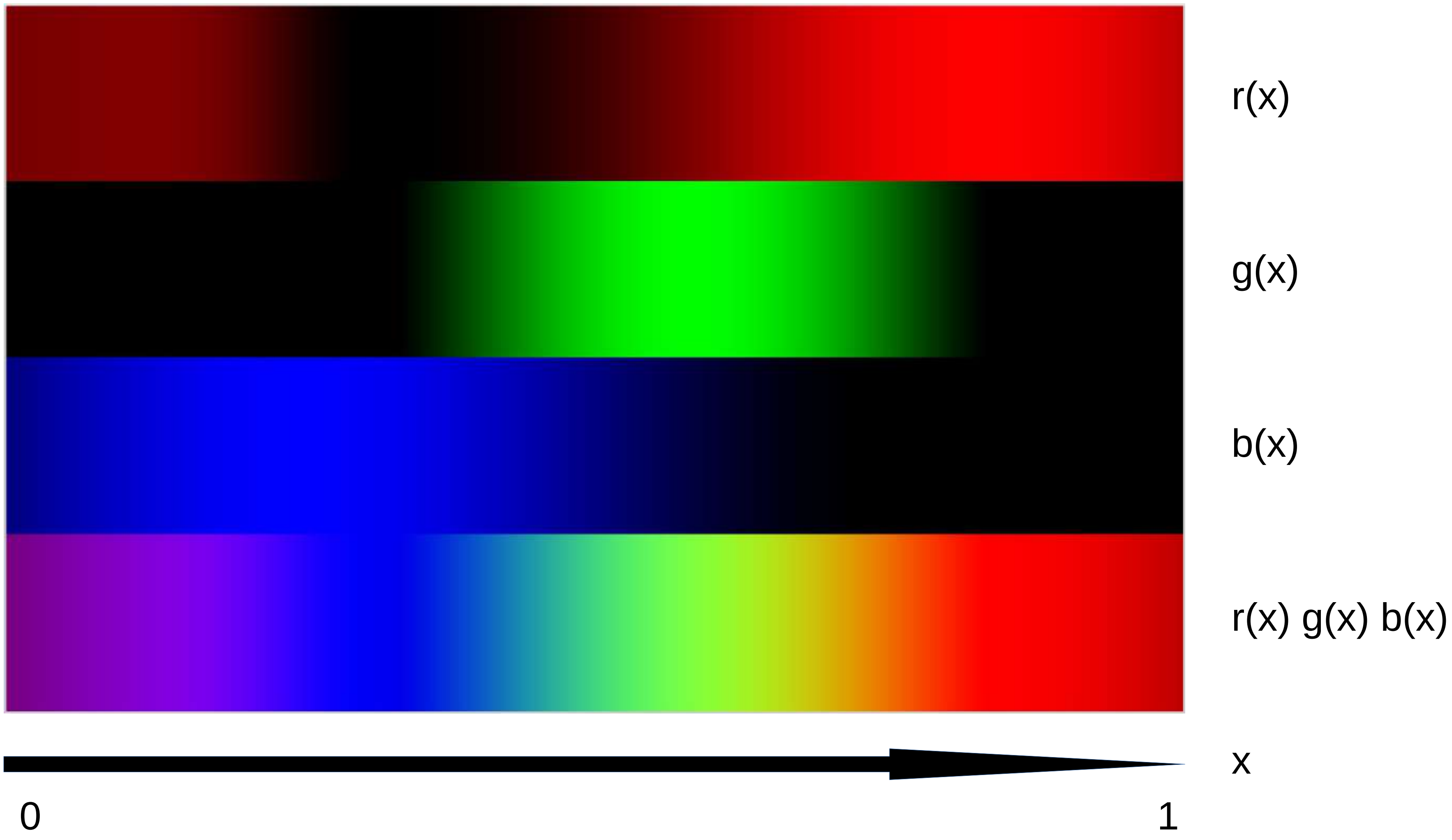


*A and B from Damon, Stoudt, McFarland, *The Human Body in Equipment Design*. C through H from *Human Factors Engineering*.

Human dimension & interior space ; Chapitre B Human Dimension / Anthropometric Tables

Annexe

Couleur



Annexe

Couleur

```
let pi = Float.pi

let r x =
    let r = x *. pi *. 2. in
    if r >= pi *. 4. /. 6. then int_of_float ((cos (r +. pi *. 2. /. 6.) +. 1.) *. 255. /. 2.)
    else begin
        if r >= 3. *. pi /. 5.
        then 0
        else let tmp = cos (r *. r -. pi /. 6.) in int_of_float ((tmp +. 1.) *. 131. /. 2.)
    end

let g x =
    let r = x *. pi *. 2. in
    if r >= pi /. 3. then let tmp = sin (r -. 4. *. pi /. 6.) in int_of_float ((Float.pow tmp 1.2) *. 255.)
    else 0

let b x =
    let r = x *. pi *. 2. in
    if r >= pi +. pi /. 2. then 0
    else int_of_float ((sin r +. 1.) *. 255. /. 2.)
```

Annexe

Code multi-agents sur grille 01

```
open Graphics
open Couleur

type case = Obstacle | Exit | Empty | People of int

type plateau = case array array

exception Solution

type etat = Traite | Traitement | Inconnu


let print_pere (p : (int * int) array array) : unit =
    for j = 0 to Array.length p -1 do
        for i = 0 to Array.length p.(0) -1 do
            match p.(j).(i) with
            | (x, y) when x = -1 && y = -1 -> Printf.printf "|  "
            | (x, y) when x = i-1 && y = j -> Printf.printf "|<"
            | (x, y) when x = i+1 && y = j -> Printf.printf "|>"
            | (x, y) when x = i && y = j-1 -> Printf.printf "|^"
            | _   -> Printf.printf "|v"
        done;
        Printf.printf "|\n";
    done;
    Printf.printf "\n";
    flush stdout
```

Annexe

Code multi-agents sur grille 02

```
let print_graph (p : plateau) c prop : unit =
    let n = Array.length p.(0) -1 in
    for j = 0 to Array.length p -1 do
        for i = 0 to n do
            set_color (rgb 150 150 150);
            draw_rect  (i*prop) ((n-j-1)*prop) prop prop;
            (match p.(j).(i) with
            | Obstacle -> set_color (rgb 0 0 0)
            | People n -> set_color c.(n)
            | _ -> set_color (rgb 250 250 250));
            fill_rect (i*prop+1) ((n-j-1)*prop+1) (prop-2) (prop-2)
        done;
    done
```

Annexe

Code multi-agents sur grille 03

```
let print_graph_chemin (p : plateau) c prop acoord achemin : unit =
    let n = Array.length p.(0) -1 in
    for j = 0 to Array.length p -1 do
        for i = 0 to n do
            set_color (rgb 150 150 150);
            draw_rect  (i*prop) ((n-j-1)*prop) prop prop;
            (match p.(j).(i) with
            | Obstacle -> set_color (rgb 0 0 0)
            | People n -> set_color c.(n)
            | _ -> set_color (rgb 250 250 250));
            fill_rect (i*prop+1) ((n-j-1)*prop+1) (prop-2) (prop-2)
        done;
    done;
    set_color (rgb 0 0 0);
    let t1 = 10 in
    (*let t2 = 4 in*)
    for i = 0 to Array.length achemin -1 do
        let xt = ref (fst acoord.(i)) in
        let yt = ref (snd acoord.(i)) in
        List.iter (fun (x, y) ->
                set_color (rgb 0 0 0);
                fill_rect ((min !xt x) * prop + prop/2 - t1/2) ((n-1-(max !yt y)) * prop + prop/2 - t1/2)
                        (t1 + (abs (!xt - x)) * prop) (t1 + (abs (!yt - y)) * prop);
                xt := x;
                yt := y) achemin.(i)
    done
```

Annexe

Code multi-agents sur grille 04

```
let valide (p : plateau) (x : int) (y : int) : bool =
    p.(y).(x) == Empty || p.(y).(x) == Exit

let mouvement (x : int) (y : int)  =
    [(x+1,y); (x-1,y); (x,y+1); (x,y-1)]

let mouvement_valide (p : plateau) (pp : etat array array) (x : int) (y : int) : (int * int) list =
    List.filter
        (fun (x,y) ->
            x >= 0 && y>=0 && y< Array.length p && x < Array.length p.(0) && (valide p x y) && pp.(y).(x) == Inconnu)
        (mouvement x y)

let trouve_chemin pere xe ye x y =
    let chemin = ref [] in
    (try
        while (!xe,!ye) <> (x, y) do
            chemin := (!xe, !ye) :: !chemin;
                let x_tmp, y_tmp = pere.(!ye).(!xe) in
                xe := x_tmp;
                ye := y_tmp
        done;
    with
        |Invalid_argument _ -> chemin := List.init (Array.length pere * Array.length pere.(0)) (fun _ -> (0, 0)));
    !chemin
```

Annexe

Code multi-agents sur grille 05

```
let dis_to_exit ld (a,b) =
    let minab = ref (Float.infinity) in
    List.iter (fun (xd,yd) ->
            let tmp1 = sqrt (float_of_int ((xd-a)*(xd-a) + (yd-b)*(yd-b))) in
            if tmp1 < !minab then minab := tmp1) ld;
    !minab

let enleve_min l =
    let mini = List.fold_left (fun (a,x) (b,y) -> if a = min a b then (a,x) else (b,y))
                    (Float.infinity, (0,0)) l in
    let rec aux l =
        match l with
        | [] -> []
        | t::q -> if t = mini then q else t :: (aux q) in
    (snd mini, aux l)
```

Annexe

Code multi-agents sur grille 06

```
let resolution (p : plateau) (x : int) (y : int) (ld : (int * int) list) : (int* int) list=
    let pere = Array.make_matrix (Array.length p) (Array.length p.(0)) (-1,-1) in
    try
        let map_deplacement = Array.make_matrix (Array.length p) (Array.length p.(0)) Inconnu in
        let f = ref [(dis_to_exit ld (x,y), (x,y))] in
        let n = List.length ld in
        let m = ref 0 in
        map_deplacement.(y).(x) <- Traitement;
        while !f <> [] do
            let tmp = enleve_min !f in
            f := snd tmp;
            let i,j = fst tmp in
            if map_deplacement.(j).(i) == Traitement
            then begin
                if p.(j).(i) = Exit then begin
                    m := !m +1;
                    if !m = n
                    then
                        raise Solution
                end;

                map_deplacement.(j).(i) <- Traite;
                List.iter (fun (ip, jp) ->
                        f := (dis_to_exit ld (ip, jp), (ip, jp)) :: !f;
                        if map_deplacement.(jp).(ip) == Inconnu
                        then begin
                            pere.(jp).(ip) <- (i, j);
                            map_deplacement.(jp).(ip) <- Traitement end)
                            (mouvement_valide p map_deplacement i j)
            end
        done;
        if !m > 0 then raise Solution
        else begin
            []
        end
```

Annexe

Code multi-agents sur grille 07

```
    with Solution ->
        print_pere pere;
        let rep = ref (List.init (Array.length p * Array.length p) (fun _ -> (0, 0))) in
        List.iter (fun (xd,yd) ->
            let xe, ye = ref xd, ref yd in
            let tmp = trouve_chemin pere xe ye x y in
            if List.length tmp < List.length !rep then rep := tmp) ld;
        !rep

let init_matrix (leny : int) (lenx : int) (f : (int * int) -> 'a) : 'a array array =
    let rep = Array.make_matrix leny lenx (f (0, 0)) in
    for i = 0 to Array.length rep -1 do
        for j = 0 to Array.length rep.(0) -1 do
            rep.(i).(j) <- f (j,i)
        done
    done;
    rep

let randomarray n max =
    if max < n then failwith "max < n";
    let rep = Array.make n (-1) in
    for i = 0 to n-1 do
        let tmp = ref (Random.int max) in
        while Array.exists (fun i -> i = !tmp) rep do
            tmp := Random.int max
        done;
        rep.(i) <- !tmp
    done;
    rep
```

Annexe

Code multi-agents sur grille 08

```
let _ =
    Random.init 90;
    let my = 20 in
    let mx = my +1 in
    let p = init_matrix my mx (fun (x,y) -> match x, y with
                                                    |_, _ when x = mx-1 && y < my/2 -1 -> Obstacle
                                                    |_, _ when x = mx-1 && y > my/2 +1 -> Obstacle
                                                    |_, _ -> Empty) in
    let ld = [(mx-1, my/2); (mx-1, my/2 -1); (mx-1, my/2 +1)] in
    List.iter (fun (xd, yd) -> p.(yd).(xd) <- Exit) ld;
    let n = 90 in
    let tmp = randomarray n (my*my) in
    let acoord = Array.init n (fun i -> tmp.(i)/my, tmp.(i) mod my) in
    let apeople = Array.init n (fun i -> i) in
    let acouleur = Array.init n (fun i ->
                                let c = float_of_int (tmp.(i)) /. float_of_int (my*my) in
                                rgb (r c) (g c) (b c)) in
    let achemin = Array.init (Array.length acoord)
                            (fun i -> resolution p (fst acoord.(i)) (snd acoord.(i)) ld) in
    let plateau = Array.init (Array.length p) (fun i -> Array.copy p.(i)) in
    for i = 0 to n-1 do
        p.(snd acoord.(i)).(fst acoord.(i)) <- People apeople.(i)
    done;

    let prop_i = 50 in
    open_graph (Printf.sprintf " %dx%d" (prop_i*mx) (prop_i*my));
    set_window_title "test";
    print_graph_chemin p acouleur prop_i acoord achemin;
```

Annexe

Code multi-agents sur grille 09

```
while Array.fold_left (+) 0 apeople <> -n do
    temp := !temp -1;
    let start_time = Sys.time () in
    for i = 0 to n-1 do
        if apeople.(i) <> -1
        then begin
            (match achemin.(i) with
            | _ when List.mem acoord.(i) ld ->
                p.(snd acoord.(i)).(fst acoord.(i)) <- Exit;
                apeople.(i) <- -1
            | [] -> achemin.(i) <- resolution plateau (fst acoord.(i)) (snd acoord.(i)) ld;
                if achemin.(i) <> []
                then begin
                    let nx, ny = List.hd achemin.(i) in
                    if valide p nx ny
                    then begin
                        p.(snd acoord.(i)).(fst acoord.(i)) <- Empty;
                        p.(ny).(nx) <- People apeople.(i);
                        achemin.(i) <- List.tl achemin.(i);
                        acoord.(i) <- (nx,ny)
                    end
                end
            | (x,y)::q when valide p x y ->
                p.(snd acoord.(i)).(fst acoord.(i)) <- Empty;
                p.(y).(x) <- People apeople.(i);
                achemin.(i) <- q;
                acoord.(i) <- (x,y)
```

Annexe

Code multi-agents sur grille 10

```
                | (x,y)::_ ->
                    plateau.(y).(x) <- Obstacle;
                    achemin.(i) <- resolution plateau (fst acoord.(i)) (snd acoord.(i)) ld;
                    plateau.(y).(x) <-Empty;
                    if achemin.(i) <> []
                    then
                        let nx, ny = List.hd achemin.(i) in
                        if valide p nx ny
                        then begin
                            p.(snd acoord.(i)).(fst acoord.(i)) <- Empty;
                            p.(ny).(nx) <- People apeople.(i);
                            achemin.(i) <- List.tl achemin.(i);
                            acoord.(i) <- (nx,ny)
                        end
                )
            end
        done;
        while Sys.time () < start_time +. 0.5 do () done;
        print_graph p acouleur prop_i
    done;
    close_graph ()
```

Annexe

Code raytracing 01

```
let pi : float = Float.pi

let rec float_mod orient =
    if orient > pi then float_mod (orient -. 2. *. pi)
    else if orient < 0. -. pi then float_mod (orient +. 2. *. pi)
    else orient

let dist_euclidienne (x,y) (x',y') =
   sqrt ((x-.x')*.(x-.x') +. (y-.y')*.(y-.y'))


let regarde_bonne_dir dx (dy : float) (o : float) : bool =
    ((dx >= 0. && cos o >= 0.) || (dx <= 0. && cos o <= 0.)) &&
    ((dy >= 0. && sin o >= 0.) || (dy <= 0. && sin o <= 0.))

let intersection_droite_rect a b o x y rep xp yp rp obji (dmax : float) (change : bool) : bool =
    let c, d, e, f = (fst (fst obji)), (snd (fst obji)), (fst (snd obji)), (snd (snd obji)) in
    let tmp_xy = (!x, !y) in
    (*
    f   .-------*
        |       |
        |  obj  |
        |       |
    d   *-------.

        c       e
    *)
```

Annexe

Code raytracing 02

```
(*verification de l'intersection avec les segments verticaux*)
(*verif de l'intersection
    verif que l'intersection se face du bon cote (droite si l'humain regarde a droite et gauche si non)
    verif que l'intersection se face sur la bonne hauteur (haut si l'humain regarde en haut et bas si non) *)
if (min d f) <= a *. c +. b && a *. c +. b <= (max d f) && regarde_bonne_dir (c-.xp) (a *. c +. b -. yp) o &&
     dist_euclidienne (c,a*. c +. b) (xp, yp) > rp && dist_euclidienne (c,a*. c +. b) (xp, yp) < dmax
then begin
    if (!x <> None &&
        dist_euclidienne (c,a*. c +. b) (xp, yp) <= dist_euclidienne (Option.get !x, Option.get !y) (xp, yp))
        || !x = None
    then begin
        x := Some c;
        y := Some (a *. c +. b);
        rep := Some obji
    end
end;
if (min d f) <= a *. e +. b && a *. e +. b <= (max d f) && regarde_bonne_dir (e -. xp) (a *. e +. b -. yp) o &&
    dist_euclidienne (e,a*. e +. b) (xp, yp) > rp && dist_euclidienne (e,a*. e +. b) (xp, yp) < dmax
then begin
    if (!x <> None &&
        dist_euclidienne (e,a*. e +. b) (xp, yp) <= dist_euclidienne (Option.get !x, Option.get !y) (xp, yp))
    || !x = None
    then begin
        x := Some e;
        y := Some (a *. e +. b);
        rep := Some obji
    end
end;
```

Annexe

Code raytracing 03

```
(*verification de l'intersection avec les segments horisontaux*)
if (min c e) <= (d -. b) /. a && (d -. b) /. a <= (max c e) && regarde_bonne_dir ((d -. b) /. a -. xp) (d -. yp) o &&
    dist_euclidienne ((d -. b) /. a,d) (xp, yp) > rp && dist_euclidienne ((d -. b) /. a,d) (xp, yp) < dmax
then begin
    if (!x <> None &&
        dist_euclidienne ((d -. b) /. a,d) (xp, yp) <= dist_euclidienne (Option.get !x, Option.get !y) (xp, yp))
        || (!x = None && !y = None)
    then begin
        x := Some ((d -. b) /. a);
        y := Some d;
        rep := Some obji
    end
end;
if (min c e) <= (f -. b) /. a && (f -. b) /. a <= (max c e) && regarde_bonne_dir ((f -. b) /. a -. xp) (f -. yp) o &&
    dist_euclidienne ((f -. b) /. a,f) (xp, yp) > rp && dist_euclidienne ((f -. b) /. a,f) (xp, yp) < dmax
then begin
    if (!x <> None &&
        dist_euclidienne ((f -. b) /. a,f) (xp, yp) <= dist_euclidienne (Option.get !x, Option.get !y) (xp, yp))
        || (!x = None && !y = None)
    then begin
        x := Some ((f -. b) /. a);
        y := Some f;
        rep := Some obji
    end
end;
let tmp = tmp_xy <> (!x, !y) in
if tmp && not change
then begin
    x := fst tmp_xy;
    y := snd tmp_xy
end;
tmp
```

Annexe

Code raytracing 04

```
let raytracing_vecteur (orient : float) (xp : float) (yp : float) (rp : float) (dmax : float)
                           (obj : ((float * float) * (float * float)) list) =
    (*orient = angle du rayon sur le cercle trigo
        (xp, yp) = position a partir de la quelle le rayon est tire
        rp = rayon a partir du quel les colition sont possible
        obj = liste des objets (rectangle) sur le terrain *)
    let o = float_mod orient in
    let a = tan o in
    let b = yp -. a *. xp in
    let rep = ref (Some ((0., 0.), (0., 0.))) in
    let x = ref (Some (xp +. dmax *. cos o)) in
    let y = ref (Some (yp +. dmax *. sin o)) in
    (*Printf.printf "orient = %f;\ta = %f;\tb = %f" o a b;*)
    let rec aux obj a b =
        match obj with
        |[] -> ()
        |obji::q ->
            ignore (intersection_droite_rect a b o x y rep xp yp rp obji dmax true);
            aux q a b
        in
    aux obj a b;
    if !x = None || Option.get !x = Float.infinity || Option.get !x = Float.neg_infinity then begin
        failwith "raytracing error"
    end;
    !rep, (Option.get !x, Option.get !y)
```

Annexe

Code 2D 01

```
open Graphics
open Bvh_tree
open Couleur

let dt : float = 1.0 /. 50.0
let dmax : float = 8. (*m*)
let k : float = 1000.

let maxX : float = 22. (*largeur de la carte en metre*)
let maxY : float = 20. (*hauteur de la carte en metre*)
let maxX_i : int = (int_of_float maxX)
let maxY_i : int = (int_of_float maxY)
let prop_i : int = 1000 / maxX_i (*affichage proportionnel 1 metre = prop pixel*)
let prop : float = (float_of_int prop_i)
let vitesse_max_humain : float = 8.3 (*m.s-1*)
let acceleration_max_humain : float = 2. (*m.s-2*)

type people = {
    largeur : float; (*en metre represente la largeur de la personne epaule droite à epaule gauche*)
    epaisseur : float; (*en metre represente l'epaisseur de la personne la profondeur du corp*)
    rayon : float; (*en metre*)
    masse : float; (*en kilogramme represente la masse de la personne*)
    allure_voulue : float; (*en m/s*)
    pos_voulue : float * float; (*en m, m*)
    mutable position : float * float; (*en m, m*)
    mutable allure : float; (*en m/s*)
    mutable direction : float; (*en radiant*)
}
```

Annexe

Code 2D 02

```
let people_init (l :float) (ep:float) (m:float) (allure : float) (pos: float * float) (allure_voulue:float)
                (pos_voulue : float * float) (dir:float) : people =
    {
        largeur = l;
        epaisseur = ep;
        rayon = dist_euclidienne (l, ep) (0., 0.) /. 2.;
        masse = m; allure_voulue = allure_voulue;
        pos_voulue = pos_voulue;
        position = pos;
        allure = allure;
        direction = dir
    }

let set_allure (p : people) a = p.allure <- min a vitesse_max_humain

let set_dir (p : people) d =  p.direction <- float_mod d

let print_people (lp : people list) lcouleur : unit=
    let rec aux lp  lcouleur j n =
        match lp, lcouleur with
        |p::qp, couleur::qcouleur ->
            (let x, y = p.position in
            set_color couleur;
            fill_circle (int_of_float (x *. prop)) (int_of_float (y *. prop)) (int_of_float(p.rayon *. prop));
            set_color (rgb 0 0 0);
            moveto (int_of_float (x *. prop)) (int_of_float (y *. prop));
            lineto (int_of_float ((x +. cos p.direction *. p.rayon) *. prop))
                    (int_of_float ((y +. sin p.direction *. p.rayon) *. prop));
            aux qp qcouleur (j +. 1.) n)
        |_, _ -> () in
    aux lp lcouleur 0. (float_of_int (List.length lp))
```

Annexe

Code 2D 03

```
let random_diff0 f =
    let temp = ref (Random.float f) in
    while !temp < 0.000001 do
        temp := Random.float f
    done;
    !temp

let set_pos lp t =
    let rec aux lp t (acc : ((float * float) * (float * float)) list) : ((float * float) * (float * float)) list =
        match lp with
        |[] -> List.rev acc
        |p::q ->
            let dis_p = p.allure *. t in
            let x, y = p.position in
            let newx = x +. (cos p.direction)*.dis_p in
            let newy = y +. (sin p.direction)*.dis_p in
            p.position <- (newx, newy);
            let a1 = asin (p.largeur /. (2. *. p.rayon)) in
            let a2 = pi/.2. +. asin (p.epaisseur /. (2. *. p.rayon)) in
            aux q t (
                ((newx, newy +. (sin (float_mod (p.direction -. a2))) *. p.rayon),
                        (newx +. (cos (float_mod (p.direction -. a2))) *. p.rayon, newy))::
                ((newx, newy -. (sin (float_mod (p.direction -. a1))) *. p.rayon),
                        (newx -. (cos (float_mod (p.direction -. a1))) *. p.rayon, newy))::
                ((newx -. (cos (float_mod (p.direction +. a2))) *. p.rayon, newy),
                        (newx, newy -. (sin (float_mod (p.direction +. a2))) *. p.rayon))::
                ((newx +. (cos (float_mod (p.direction +. a1))) *. p.rayon, newy),
                        (newx, newy +. (sin (float_mod (p.direction +. a1))) *. p.rayon))::acc) in
    aux lp t []
```

Annexe

Code 2D 04

```
let min_abs a b = if min (Float.abs a) (Float.abs b) = Float.abs a then a else b


let print_bvh_tree a =
    let rec aux a =
        match a with
        |Nil -> ()
        |Noeud (g,o,d) ->
            set_color (rgb 65 0 125);
            fill_rect
                (int_of_float ((fst (fst o)) *. prop))
                (int_of_float ((snd (fst o)) *. prop))
                (abs(int_of_float (((fst (fst o))-.(fst (snd o))) *. prop)))
                (abs(int_of_float (((snd (fst o))-.(snd (snd o))) *. prop)));
            aux g;
            aux d;
        |Feuille o ->
            set_color (rgb 125 0 0);
            fill_rect
                (int_of_float ((fst (fst o)) *. prop))
                (int_of_float ((snd (fst o)) *. prop))
                (abs(int_of_float (((fst (fst o))-.(fst (snd o))) *. prop)))
                (abs(int_of_float (((snd (fst o))-.(snd (snd o))) *. prop))) in
    aux a
```

Annexe

Code 2D 05

```
let print_obj l =
    let rec aux l =
        set_color (rgb 0 0 0);
        match l with
        |[] -> ()
        |o::q ->
            let x = min (fst (fst o)) (fst (snd o)) in
            let y = min (snd (fst o)) (snd (snd o)) in
            let w = max (fst (fst o)) (fst (snd o)) -. x in
            let h = max (snd (fst o)) (snd (snd o)) -. y in
            fill_rect (int_of_float (x *. prop)) (int_of_float (y *. prop))
                        (int_of_float (w *. prop)) (int_of_float (h *. prop));
            aux q
        in
    aux l


let rm lp lc =
    let rec aux lp lc acclp acclc accn =
        match lp, lc with
        | p::qp, c::qc ->
            (*if dist_euclidienne p.position p.pos_voulue < p.rayon*)
            if fst p.position >= fst p.pos_voulue
            then
                aux qp qc acclp acclc accn
            else
                aux qp qc (p::acclp) (c::acclc) (1+accn)
        | _, _ -> ((acclp, acclc), accn) in
    aux lp lc [] [] 0
```

Annexe

Code 2D 06

```
let new_orient (l : ('a * 'b * float) list) (dmax : float) (alpha0 : float)=
    let rec aux l min acc =
        match l with
        | [] -> acc
        | (dist, _, alpha)::q ->
            let tmp = dmax *. dmax +. dist *. dist
                -. 2. *. dmax *. dist *. cos (alpha0 -. alpha)
                +. dmax *. Float.abs (tan ((alpha0 -. alpha) /. 2.)) in
            if tmp < min
            then aux q tmp alpha
            else aux q min acc in
    aux l Float.infinity 0.

let changement_direction_allure (lp : people list) (ll : (float * (float * float) * float) list list)
        (obj : ((float * float) * (float * float)) list) (completobj : ((float * float) * (float * float)) list) : unit =
    (*  lp : liste des personne presente
        ll : liste de (liste des point vu par une personne) par personne
        obj : liste des objects sur le terrain *)
    let rec somme_fij lp ri mi posi i j accx accy =
        match lp with
        | [] -> (k *. accx /. mi, k *. accy /. mi)
        | p::qlp -> if i = j
            then
                somme_fij qlp ri mi posi i (j+1) accx accy
            else begin
                let g = ri +. p.rayon -. dist_euclidienne posi p.position in
                if g <= 0.
                then
                    somme_fij qlp ri mi posi i (j+1) accx accy
                else begin
                    let alphaij = atan2 (snd posi -. snd p.position) (fst posi -. fst p.position) in
                    somme_fij qlp ri mi posi i (j+1) (accx +. g *. cos alphaij) (accy +. g *. sin alphaij)
                end
            end
        in
```

Annexe

Code 2D 07

```
let rec somme_fiw obj ri mi xi yi (accx: float) accy =
    match obj with
    | [] -> (k *. accx /. mi, k *. accy /. mi)
    | ((x1, y1), (x2, y2))::qobj ->
        let gx = min x1 x2 in
        let dx = max x1 x2 in
        let by = min y1 y2 in
        let hy = max y1 y2 in
        let x = ref xi in
        let y = ref yi in
        if dx < xi then x := dx;
        if xi < gx then x := gx;
        if hy < yi then y := hy;
        if yi < by then y := by;
        let g = ri -. dist_euclidienne (xi, yi) (!x, !y) in
        if g <= 0.
        then
            somme_fiw qobj ri mi xi yi accx accy
        else begin
            let alphaij = atan2 (yi -. !y) (xi -. !x) in
            somme_fiw qobj ri mi xi yi (accx +. g *. cos alphaij) (accy +. g *. sin alphaij)
        end
    in
```

Annexe

Code 2D 08

```
let rec aux auxlp ll acc =
    match auxlp,ll with
    |p::qp,l::ql ->
        let alpha0 = 0. -. p.direction +.
            atan2 (snd p.pos_voulue -. snd p.position) (fst p.pos_voulue -. fst p.position) in
        let alpha_des = p.direction +. new_orient l dmax alpha0 in
        let (vix, viy) = p.allure *. (cos p.direction), p.allure *. (sin p.direction) in
        let tmp = raytracing_vecteur
                        alpha_des
                        (fst p.position)
                        (snd p.position)
                        p.rayon
                        (sqrt (maxX *. maxX +. maxY *. maxY))
                        completobj in
        let dh = dist_euclidienne p.position (snd tmp) in
        let v_des = min p.allure_voulue (dh /. dt) in
        let (vx_des, vy_des) = (v_des *. cos alpha_des, v_des *. sin alpha_des) in
        let (afijx, afijy) = somme_fij lp p.rayon p.masse p.position acc 0 0. 0. in
        let (afiwx, afiwy) = somme_fiw obj p.rayon p.masse (fst p.position) (snd p.position) 0. 0. in
        let vx = vix +. (vx_des -. vix) *. 0.5 +. afijx +. afiwx in
        let vy = viy +. (vy_des -. viy) *. 0.5 +. afijy +. afiwy in
        let new_allure = sqrt (vx *. vx +. vy *. vy) in
        set_allure p (new_allure);
        set_dir p (atan2 vy vx);
        aux qp ql (acc+1)
    |_,_ -> () in
aux lp ll 0
```

Annexe

Code 2D 09

```
let rec detection_pt_physique (lpeople : people list) (obj : ((float*float)*(float*float)) list)
                              : (float * (float * float) * float) list list =
    (*lpeople : liste des humains presents
        obj : tableau contenant les objets qu'il y a sur le terrain*)
    match lpeople with
    | [] -> []
    | p::q ->
        let orient_ray = ref 0. in
        let pt_vu = ref [] in
        moveto (int_of_float(fst p.position *. prop)) (int_of_float(snd p.position *. prop));
        set_color (rgb 150 25 25);
        for _ = 0 to 255 do
            let r1 = random_diff0 1. in
            let r2 = Random.float 1. in
            let z = (sqrt((-2.) *. log(r1)) /. 3.899) *. cos(2. *. pi *. r2) /. 1.348172 in
            orient_ray := asin (z);
            set_color (rgb 150 25 25);
            (match raytracing_vecteur (p.direction +. !orient_ray) (fst p.position) (snd p.position) p.rayon dmax obj with
            |Some _,(x,y) ->
                            let dist = dist_euclidienne (x,y) p.position in
                            pt_vu := (dist, (x,y), !orient_ray) :: !pt_vu
            |None,_ -> ());
            (match raytracing_vecteur (p.direction -. !orient_ray) (fst p.position) (snd p.position) p.rayon dmax obj with
            |Some _,(x,y) ->
                            let dist = dist_euclidienne (x, y) p.position in
                            pt_vu := (dist, (x,y), 0. -. !orient_ray) :: !pt_vu

            |None,_ -> ())
        done;
        !pt_vu :: (detection_pt_physique q obj)
```

Annexe

Code 2D 10

```
let randomarray n max =
    if max < n then failwith "max < array length";
    let rep = Array.make n (-1) in
    for i = 0 to n-1 do
        let tmp = ref (Random.int max) in
        while Array.exists (fun i -> i = !tmp) rep do
            tmp := Random.int max
        done;
        rep.(i) <- !tmp
    done;
    rep

let test_prog () =
    Random.init 90;
    let pause = 1000 in
    let nbboucle = ref 0 in

    open_graph (Printf.sprintf " %dx%d" (prop_i*maxX_i) (prop_i*maxY_i));
    set_window_title "Deplacement test";
    set_color (rgb 0 0 0);
    fill_rect 0 0 (maxX_i*prop_i) (maxY_i*prop_i);
    set_color (rgb 255 255 255);
```

Annexe

Code 2D 11

```
let n = ref 90 in
let tab = randomarray !n (maxY_i * maxY_i) in
let ltest = ref (List.init !n (fun i -> people_init 0.579 0.33 70.
    0. (float_of_int (tab.(i)/maxY_i) +. 0.5, float_of_int (tab.(i) mod maxY_i) +. 0.5)
    1. (maxX -. 1.5, maxY /. 2.) (Random.float (2. *. pi)))) in
let lcouleur = ref (List.init !n (fun i ->
    let c = float_of_int (tab.(i)) /. float_of_int (maxY_i * maxY_i) in
    rgb (r c) (g c) (b c))) in

let obj = [((-1., Float.neg_infinity), (0., Float.infinity));
        ((maxX, Float.neg_infinity), (maxX+.1., Float.infinity));
        ((Float.neg_infinity, -1.), (Float.infinity, 0.));
        ((Float.neg_infinity, maxY), (Float.infinity, maxY+.1.));
        (((maxX -. 2.), 0.), ((maxX -. 1.), (maxY /. 2.) -. 0.5));
        (((maxX -. 2.), (maxY /. 2.) +. 0.5), ((maxX -. 1.), maxY));] in

let people_block = ref [] in
```

Annexe

Code 2D 12

```
 while !n > 0 && !nbboucle < pause do
     let time = Sys.time() in
     set_color (rgb 250 250 250);
     fill_rect 0 0 (maxX_i*prop_i) (maxY_i*prop_i);
     set_color (rgb 0 0 0);
     moveto 0 0;
     draw_string (Printf.sprintf "time %f s" (float_of_int (!nbboucle) *. dt));

     let tmp = rm !ltest !lcouleur in
     ltest := fst (fst tmp);
     lcouleur := snd (fst tmp);
     n := snd tmp;

     people_block := set_pos !ltest dt; (*O(n)*)

     print_people !ltest !lcouleur; (*O(n)*)
     print_obj obj; (*O(m)*)

     let lo = detection_pt_physique !ltest (!people_block @ obj) in (*O(n*510*(4n+m)*)
     changement_direction_allure !ltest lo obj (!people_block @ obj); (*O(n*(4n+m +n +m)*)
     nbboucle := !nbboucle +1;
     while Sys.time () < time +. (dt *. 1.) do () done;
done;
while true do () done;
close_graph ()
```